A.C.E

Automotive Clarity Enhancer

מגיש:

אופיר נבו מיכרובסקי

ת"ז:

216763433

מנחה:

ניר דוויק

בית-ספר:

תיכון הדסים

תאריך:

03/06/2025

תוכן העניינים:

מבוא

מבוא - ייזום

תיאור כללי:

המערכת המוצעת הינה מערכת שידור וידאו בזמן אמת, הכוללת מצלמה אלחוטית המחוברת למסך אינטרנטי, עם פונקציונליות מתקדמת של הוספת כתוביות לדיאלוג המתועד. המסך המרכזי יציג את המצלמה, יחד עם תמלול מדויק של המדובר.
מוצר זה מיועד לשמש כרכיב תוכנתי חיוני במערכת עזר שמיעה אישית המותקנת ברכבים. פרויקט זה נולד מתוך הבנה עמוקה של הקשיים הייחודיים העומדים בפני אנשים כבדי שמיעה, במיוחד בסביבה רועשת כמו רכב נוסע. **המוצר הינו תוכנתי בלבד**.

אנשים עם לקויות שמיעה מתמודדים עם אתגרים רבים בעת נסיעה ברכב, שכן הם מתקשים לראות את שפתי הנוסעים האחרים ולקרוא אותן, ומכשירי השמיעה שלהם לעיתים קרובות לא יעילים בשל רעשי המנוע והכביש. מערכת זו נועדה לספק פתרון חדשני ויעיל, שיאפשר להם להשתתף באופן פעיל בשיחות המתנהלות ברכב, וליהנות מחוויית נסיעה נעימה ומכילה יותר.

המערכת תשתמש במצלמות אינטרנט פשוטות, ותציע מנגנון תמלול אוטומטי ע"י בינה מלאכותית. דגש מיוחד יושם על עיצוב ממשק משתמש נגיש, פשוט, נוח ואינטואיטיבי, שיתאים לצרכים הייחודיים של קהל היעד.

הגדרת לקוח:

מערכת זו מיועדת בראש ובראשונה לאנשים כבדי שמיעה, מכל קבוצות הגיל, המעוניינים להשתתף באופן מלא בשיחות המתנהלות ברכב. המערכת תוכננה במיוחד עבור משפחות ואנשים המרבים בנסיעות משותפות, ומעוניינים ליצור סביבה נעימה ומכילה עבור כלל הנוסעים. בנוסף, המערכת עשויה להתאים גם לאנשים הסובלים מקשיי שמיעה זמניים, הורים חדשים בתור מוניטור לתינוק, או לאנשים המעוניינים לשפר את חוויית הנסיעה שלהם באמצעות טכנולוגיה מתקדמת.

חשוב לציין כי המערכת מתאימה לסוגים שונים של רכבים, החל מרכבים פרטיים ועד לרכבי שטח. אנו שואפים להנגיש את הטכנולוגיה הזו לכמה שיותר אנשים, על מנת לאפשר להם ליהנות מנסיעה בטוחה, נעימה ומעשירה יותר.

הגדרת יעדים / מטרות:

מטרת העל של הפרויקט היא לשפר את איכות חייהם של אנשים כבדי שמיעה באמצעות טכנולוגיה מתקדמת. המערכת שואפת להעניק להם את היכולת להשתתף באופן פעיל בשיחות המתנהלות ברכב, דבר המהווה אתגר משמעותי עבורם כיום.

המטרות המרכזיות של המערכת כוללות:

- **שיפור הנגישות :**להפוך את הנסיעה ברכב לנגישה יותר עבור אנשים כבדי שמיעה.
- **עידוד השתתפות :**לאפשר לאנשים כבדי שמיעה להשתתף באופן מלא בשיחות ובפעילויות המתנהלות ברכב.
- **העלאת הביטחון העצמי :**להגביר את ביטחונם העצמי של אנשים כבדי שמיעה בעת נסיעה ברכב.
- **הפחתת תחושת הבדידות :**למנוע תחושת בדידות וניכור בקרב אנשים כבדי שמיעה במהלך נסיעות.
- **יצירת סביבה מכילה :**לתרום ליצירת סביבה נעימה ומכילה עבור כלל הנוסעים ברכב.

בעיות, תועלות וחסכונות:

מערכת זו נועדה לתת מענה למספר בעיות עיקריות:

- **קושי בתקשורת:** אנשים כבדי שמיעה מתקשים להשתתף בשיחות ברכב, במיוחד כאשר הם לא יכולים לראות את שפתי הדוברים.
- **רעש סביבתי:** רעשי מנוע, כביש וסביבה חיצונית מקשים על השימוש במכשירי שמיעה.
- **בידוד חברתי:** קושי בהשתתפות בשיחות גורם לבידוד חברתי ולתחושת ניכור.

התועלות הצפויות מהמערכת:

- **שיפור בתקשורת:** המערכת תאפשר לאנשים כבדי שמיעה להבין את הנאמר בצורה טובה יותר.
- **השתתפות פעילה:** המערכת תעודד השתתפות פעילה בשיחות ותמנע תחושת בדידות.
- **נוחות ובטיחות:** המערכת תשפר את חוויית הנסיעה ותגביר את תחושת הביטחון.
- **נגישות:** המערכת תהיה נגישה וקלה לשימוש עבור אנשים עם מוגבלויות שונות.

החסכונות הצפויים:

- **חסכון בזמן:** שיפור התקשורת ימנע אי הבנות ויחסוך זמן.
- **חסכון במאמץ:** המערכת תפחית את המאמץ הנדרש להבנת הנאמר.
- **חסכון רגשי:** המערכת תתרום לרווחה רגשית ולמניעת תסכול.

סקירת פתרונות קיימים:

בעת בחינת פתרונות קיימים עבור אנשים כבדי שמיעה ברכבים, נמצאו מספר גישות שונות. כל אחת מהגישות הללו מציעה מענה מסוים לאתגרים העומדים בפני אנשים כבדי שמיעה, אך לכל אחת גם מגבלות וחסרונות. הטבלה הבאה משווה בין הפתרונות הקיימים למערכת המוצעת, ומדגישה את היתרונות והחסרונות של כל אחד מהם:

מאפיין	המערכת המוצעת	מערכות כתוביות	אפליקציות תמלול	מכשירי שמיעה
עיקרון פעולה	שילוב וידאו ותמלול	הצגת כתוביות	תמלול בזמן אמת	הגברת עוצמת הקול
יתרונות	אינטגרציה, אוטומציה, נגישות, התאמה אישית	נוחות, קריאות	זמינות, עלות נמוכה	שיפור שמיעה, ניידות
חסרונות	עלות ראשונית	לא מותאם לשימוש ברכב	לא נוח לשימוש ברכב, דורש הפעלה ידנית	מוגבל ביעילות בסביבה רועשת, דורש התאמה אישית
התאמה לרכב	מותאמת	לא מותאמת	לא מותאמת	מוגבלת

נוחות שימוש	נוח	לא נוח	לא נוח	משתנה
נגישות	גבוהה	משתנה	משתנה	משתנה

לסיכום, המערכת המוצעת מהווה פתרון חדשני ויעיל עבור אנשים כבדי שמיעה ברכבים, ומהווה שיפור משמעותי על פני הפתרונות הקיימים.

<u>סקירת טכנולוגיות הפרויקט:</u>

פרויקט זה מתמקד בפיתוח רכיב התוכנה של מערכת עזר שמיעה לרכב, ולכן הטכנולוגיות שנבחרו ממלאות תפקיד קריטי בהצלחת הפרויקט. הטכנולוגיות הללו נבחרו בקפידה על מנת להבטיח את הפונקציונליות, היעילות והאמינות הגבוהה ביותר של המערכת.

להלן סקירה של הטכנולוגיות העיקריות שיילקחו בחשבון בפיתוח המערכת:

- **שפת תכנות :**פייתון. פייתון נבחרה בשל היותה שפה ורסטילית, קריאה ובעלת ביצועים גבוהים, המתאימה לפיתוח מערכות מורכבות כמו זו.
- **ספריית עיבוד תמונה:** OpenCV (Open Source Computer Vision Library). ספרייה זו מספקת כלים מקיפים לעיבוד וידאו בזמן אמת, החל מלכידת וידאו ממצלמות או קבצים, דרך עיבוד תמונה וזיהוי אובייקטים, ועד להצגת וידאו.
- **מערכת זיהוי דיבור ותמלול:** Google Cloud Speech-To-Text. מערכת זו מציעה תמלול אוטומטי בזמן אמת, זיהוי שפות והתאמה אישית, אשר חיוניים לפעילות המערכת.
- **מסד נתונים:** SQLite. מסד נתונים זה ישמש לאחסון וניהול נתונים, כגון הגדרות משתמש ונתוני תמלול.
- **פלטפורמת פיתוח**: JetBrains Pycharm Community Edition. פלטפורמה זו נבחרה בשל היותה סביבת פיתוח נוחה ויעילה, המאפשרת פיתוח חלק ומהיר.

חשוב להדגיש כי הפרויקט מתמקד במימוש **היבט התוכנה בלבד** של המערכת. לכן, לא תתבצע בפועל התקנה של המערכת ברכבים. הפיתוח יתמקד ביצירת סימולציה של המערכת, שתדגים את פעולתה ותאפשר הערכה של יעילותה.

במהלך הפיתוח, יושם דגש מיוחד על אופטימיזציה של ביצועים, עיצוב ממשק משתמש נגיש ואבטחת מידע. אלו נושאים בעלי חשיבות עליונה, אשר יבטיחו את איכותה ויעילותה של המערכת.

תיחום הפרויקט:

פרויקט זה מתמקד בפיתוח רכיב התוכנה של מערכת עזר שמיעה לרכב. במסגרת זו, הפרויקט נוגע במספר תחומים עיקריים:

- **עיבוד וידאו:** הפרויקט עוסק בלכידה, עיבוד והצגה של זרם וידאו ממספר מצלמות.
- **תקשורת אלחוטית:** הפרויקט כולל פיתוח מנגנון תקשורת **אלחוטית** בין המצלמות למסך המרכזי, לצורך שידור וידאו בזמן אמת.
- **זיהוי דיבור ותמלול:** הפרויקט משלב מערכת זיהוי דיבור ותמלול בזמן אמת, לצורך המרת דיבור לכתוביות.
- **פיתוח תוכנה :**הפרויקט כולל תכנון, פיתוח ובדיקה של רכיבי התוכנה השונים.
- **עיצוב ממשק משתמש:** הפרויקט שם דגש על עיצוב ממשק משתמש נגיש ואינטואיטיבי, המתאים לצרכים של אנשים כבדי שמיעה.

חשוב להדגיש כי הפרויקט **אינו** עוסק בתחומים הבאים:

- **התקנה פיזית:** הפרויקט אינו כולל התקנה פיזית של המערכת ברכבים.
- **חומרה:** הפרויקט אינו עוסק בפיתוח או רכישה של רכיבי חומרה, כגון מצלמות או מסכים.
- **אינטגרציה עם מערכות רכב:** הפרויקט אינו כולל אינטגרציה עם מערכות קיימות ברכב (כגון מערכת שמע או מערכת מולטימדיה).
- **בדיקות בשטח:** הפרויקט אינו כולל בדיקות של המערכת בתנאי שטח, כגון נסיעות מבחן.
- **מערכות הפעלה:** הפרויקט אינו עוסק במערכות הפעלה מכיוון שהפרויקט הולך לרוץ על מערכת ווינדוס עד שתהיה אפשרות להתקין את זה על חומרה.

אפיון

תיאור של המערכת:

המערכת שפותחה הינה מערכת תוכנה שמטרתה לסייע לאנשים כבדי שמיעה להשתתף בשיחות המתנהלות ברכב. היא עושה זאת על ידי לכידת וידאו ממצלמה אלחוטית ברכב, עיבוד התמונה והצגתה על מסך אינטרנטי, ותמלול בזמן אמת של השיחה.

המערכת פועלת באמצעות מספר מחשבי Windows אשר יריצו את התוכנה. מחשב אחד ישלח אודיו ווידאו למחשב שרת ייעודי. שרת זה ימיר את האודיו לכתוביות וישלח אותן ויעלה אותן לשרת ווב שיציג את התמלול עם הוידאו באתר, אשר מציג את השידור. על גבי שידור זה יוספו הכתוביות בזמן אמת.

פירוט יכולות:

- **לכידת ושידור וידאו ממצלמות:** המערכת מסוגלת לקבל ולעבד זרם וידאו ממספר מצלמות אלחוטיות הממוקמות ברכב.
- **עיבוד תמונה:** המערכת מבצעת עיבוד תמונה של זרם הווידאו, כגון התאמת גודל התמונה והצגת התמונות על המסך המרכזי.
- **תמלול בזמן אמת:** המערכת משלבת מערכת זיהוי דיבור ותמלול בזמן אמת, לצורך המרת דיבור לכתוביות המוצגות על המסך המרכזי.
- **הצגת כתוביות:** המערכת מציגה את הכתוביות על המסך המרכזי, באופן סינכרוני עם זרם הווידאו.
- **ממשק משתמש נגיש:** המערכת כוללת ממשק משתמש נגיש ואינטואיטיבי, המותאם לצרכים של אנשים כבדי שמיעה.

פירוט הבדיקות (קופסה שחורה)

במסגרת פרויקט זה, יבוצעו בדיקות קופסה שחורה אוטומטיות בתחילת ההרצה של התוכנה, על מנת להבטיח את תקינותה ומוכנותה לפעולה. להלן פירוט הבדיקות המתוכננות:

1. **בדיקות חיבוריות:**
   - **בדיקת חיבור לשרת :**
     - **מטרה :**לוודא חיבור מהיר ויציב לשרת התמלול.
     - **אופן הבדיקה :**התוכנה תנסה להתחבר לשרת.
     - **תוצאה מצופה :**חיבור מוצלח לשרת, או דיווח על שגיאת חיבור.
2. **בדיקות פונקציונליות בסיסיות:**
   - **בדיקת לכידת וידאו :**
     - **מטרה :**לוודא לכידה תקינה של וידאו מכל המצלמות המחוברות.
     - **אופן הבדיקה :**התוכנה תנסה ללכוד וידאו מכל המצלמות.
     - **תוצאה מצופה :**לכידת וידאו מוצלחת מכל המצלמות.
   - **בדיקת קליטת אודיו :**
     - **מטרה :**לוודא קליטה תקינה של אודיו מהמיקרופון המחובר.
     - **אופן הבדיקה :**התוכנה תנסה לקלוט אודיו מהמיקרופון.
     - **תוצאה מצופה :**קליטת אודיו מוצלחת מהמיקרופון.
   - **בדיקת תמלול בזמן אמת :**
     - **מטרה :**לוודא תמלול אודיו בזמן אמת בצורה מדויקת.
     - **אופן הבדיקה :**התוכנה תשלח אודיו לשרת ותבדוק את התמלול המתקבל.
     - **תוצאה מצופה :**תמלול מדויק של האודיו.
   - **בדיקת הצגת כתוביות :**
     - **מטרה :**לוודא הצגה תקינה של כתוביות על המסך.
     - **אופן הבדיקה :**התוכנה תציג כתוביות על המסך.
     - **תוצאה מצופה :**הצגה תקינה של הכתוביות, סנכרון עם הווידאו.

תכנון וניהול לו"ז לפיתוח המערכת:

לוח הזמנים הנ"ל הוא חילוק של הפרויקט ל-5 חלקים שונים של פיתוח המערכת, השלב הפרה-התחלתי, התחלתי, הבסיסי, המתקדם והסופי.

1. **השלב הפרה-התחלתי** – שלב זה יכיל הכנות של הפרויקט כולל כתיבת תבנית לפרויקט, התחלה של הלוגים ובנייה של הלוגרים. שלב זה יכיל את כל ההכנות כדי שהשלבים הבאים יוכלו להתקדם יותר ביעילות.
תוצרים: עיצוב של המערכת בנוסף לתבנית שאפשר להתבסס עליה.

2. **השלב ההתחלתי** – פיתוח של רכיבי המערכת יצירה של צ'אט שידור וידאו בסיסי לפי דרישת המערכת, יצירת מערכת משתמשים ורישום ויצירה של התמלול מוצמד לשידור.
תוצרים: קוד מקור של רכיבי המערכת, בדיקות יחידה

3. **השלב הבסיסי** – איחוד של חלקי המערכת שפותחו בשלב ההתחלתי.
תוצרים: מערכת בשלב alpha

4. **השלב המתקדם** – עיצוב של ממשק המשתמש לצורה יותר נעימה לעין כך ושיפור הביצועים ע"פ דו"ח הבדיקות ותיקון באגים סופיים.
תוצרים: מערכת מתוקנת ומשופרת.

5. **השלב הסופי** – סיכום של הפרויקט להוסיף בדיקות סופיות וסיום ספר פרויקט ומדריך משתמש ולהריץ בדיקות שטח.
תוצרים: מערכת מותקנת ומוכנה, מדריך משתמש וספר פרויקט.

שלב	לו"ז ראשוני	לו"ז בפועל
השלב הפרה-התחלתי	1/3/25	16/3
השלב ההתחלתי	15/4/25	20/5 (ללא התמלול)
השלב הבסיסי	15/5/25	טרם
השלב המתקדם	31/5/25	טרם
השלב הסופי	3/6/25	טרם

ניהול הסיכונים בפרויקט:

במהלך פיתוח הפרויקט, זוהו מספר סיכונים פוטנציאליים אשר עשויים להשפיע על הצלחת הפרויקט. להלן פירוט הסיכונים והדרכים להתמודדות עמם:

מס ' בדיקה	שם הבדיקה (שיעיד על התוכן)	מה אמורה לבדוק	איך מתכננים לבדוק (לתאר בפירוט את שלבי הבדיקה)
1	בדיקת חיבוריות - חיבור למצלמות אלחוטיות	חיבור מהיר ויציב לכל המצלמות המוגדרות	בדיקה אוטומטית של חיבור לכל המצלמות המוגדרות. דיווח על שגיאות חיבור (אם קיימות).
2	בדיקת חיבוריות - חיבור למיקרופון	זיהוי וחיבור תקין למיקרופון המוגדר	בדיקה אוטומטית של חיבור למיקרופון המוגדר. דיווח על שגיאות חיבור (אם קיימות).
3	בדיקת חיבוריות - חיבור לשרת	חיבור מהיר ויציב לשרת התמלול	בדיקה אוטומטית של חיבור לשרת. דיווח על שגיאות חיבור (אם קיימות).
4	בדיקה פונקציונלית - לכידת וידאו	לכידה תקינה של וידאו מכל המצלמות המחוברות	בדיקה אוטומטית של לכידת וידאו מכל המצלמות. וידוא איכות וידאו משתנה (רזולוציה, קצב פריימים).
5	בדיקה פונקציונלית - קליטת אודיו	קליטה תקינה של אודיו מהמיקרופון המחובר	בדיקה אוטומטית של קליטת אודיו מהמיקרופון. וידוא איכות שמע משתנה (עוצמה, רעשי רקע).
6	בדיקה פונקציונלית - תמלול בזמן אמת	תמלול אודיו בזמן אמת בצורה מדויקת	שימוש בקובץ אודיו מקומי לבדיקה בדיקה אוטומטית של התמלול המתקבל. וידוא תמיכה בשפות שונות, טיפול בהפסקות בדיבור.
7	בדיקה פונקציונלית - הצגת כתוביות	הצגה תקינה של כתוביות על המסך	בדיקה אוטומטית של הצגת כתוביות על המסך. וידוא סנכרון בין כתוביות לווידאו, אפשרויות עיצוב כתוביות.
8	בדיקת משאבי מערכת	צריכת משאבים (CPU, זיכרון (תקינה בעת הפעלת התוכנה	ניטור אוטומטי של צריכת משאבים. דיווח על חריגות.
9	בדיקת ממשק משתמש	תקינות ממשק המשתמש (כפתורים, תפריטים, תצוגה)	בדיקה אוטומטית של תקינות רכיבי הממשק.

			וידוא נגישותיות למשתמשים עם מוגבלויות.

תיאור תחום הידע

<u>יכולות בצד שרת</u>

- **טיפול בחיבורים מרובים**
  - מהות: ניהול חיבורים של לקוחות משני הסוגים (משדר וצופה) וטיפול סימולטני ויעיל בהם.
  - אוסף יכולות/פעולות נדרשות :
    - קבלת חיבורים חדשים
    - זיהוי סוג חיבור (משדר/צופה)
    - ניהול רשימת חיבורים פעילים
    - טיפול בניתוקים
  - אובייקטים נחוצים: רשימת חיבורים, מנגנון זיהוי סוג חיבור, מנגנון ניהול משאבים, מנגנון ניהול משתמשים, ניהול משדרים וצופים
- **קבלת נתוני אודיו ווידאו**
  - מהות: קבלה ועיבוד של נתוני אודיו בזמן אמת מלקוח משדר.
  - אוסף יכולות/פעולות נדרשות :
    - קבלת נתוני אודיו מהלקוח
    - בדיקת תקינות הנתונים
  - אובייקטים נחוצים: מנגנון קבלה, מנגנון עיבוד אודיו ווידאו

- **תמלול טקסט מתוך אודיו**
  - מהות: המרת דיבור לתמלול טקסטואלי בזמן אמת.
  - אוסף יכולות/פעולות נדרשות :
    - זיהוי דיבור
    - המרת דיבור לטקסט
    - עיבוד תוצאות תמלול
  - אובייקטים נחוצים: מנוע זיהוי דיבור, מנגנון עיבוד טקסט

- **ניהול מאגר מידע של שיחות קודמות**
  - מהות: ניהול מאגר מידע של שיחות מתועדות.
  - אוסף יכולות/פעולות נדרשות:
    - גישה למסד נתונים
    - ניהול משתמשים (הוספה, מחיקה וקריאה)
  - אובייקטים נחוצים: מסד נתונים

<u>יכולות בצד לקוח משדר</u>

- **שליחת נתוני וידאו ואודיו**
  - מהות: לכידה ושליחה בזמן אמת של נתוני וידאו ואודיו מהמצלמה והמיקרופון לשרת.
  - אוסף יכולות/פעולות נדרשות :
    - גישה למצלמה ולמיקרופון
    - לכידת נתוני וידאו ואודיו
    - שליחה לשרת
  - אובייקטים נחוצים: ניהול קליטת וידאו, ניהול קליטת אודיו, פרוטוקול, מנהל תקשורת

<u>יכולות בצד לקוח צופה</u>

- **קבלת תמלול ווידאו**
  - מהות: קבלה והצגה של תמלול טקסט ווידאו בזמן אמת מהשרת.
  - אוסף יכולות/פעולות נדרשות :
    - קבלה של נתוני תמלול ווידאו מהשרת
    - פענוח הנתונים
  - אובייקטים נחוצים: מנגנון קבלה, מנגנון פענוח
- **הצגת וידאו עם תמלול**
  - מהות: הצגה סינכרונית של הווידאו והתמלול על המסך.
  - אוסף יכולות/פעולות נדרשות :
    - הצגת וידאו
    - הצגת תמלול
    - סנכרון בין וידאו לתמלול
  - אובייקטים נחוצים: מנגנון הצגה, מנגנון סנכרון
- **עיבוד תמונה**
  - מהות: עיבוד תמונה (התאמת גודל של התמונה למסך ע"פ כמות מצלמות).
  - אוסף יכולות/פעולות נדרשות :
    - קבלת תמונה
    - שינוי גודל תמונה
  - אובייקטים נחוצים: אין => נעשה דרך CSS

<u>יכולות כלליות</u>

- **תקשורת מוצפנת בהצפנה אסימטרית**
  - מהות: אבטחת התקשורת בין הלקוחות לשרת באמצעות הצפנה אסימטרית.
  - אוסף יכולות/פעולות נדרשות :
    - הצפנה של הודעות
    - פענוח של הודעות
    - **שימוש בספרייה חיצונית עקב שימוש בדפדפן**
  - אובייקטים נחוצים: מנגנון הצפנה אסימטרית
- **ממשק משתמש וויזואלי**
  - מהות: ממשק משתמש אינטואיטיבי המאפשר צפייה בשידור או שליחה
  - אוסף יכולות/פעולות נדרשות :
    - הצגת וידאו ותמלול
    - שידור וידאו
    - כניסת ויצירת משתמש
    - בחירה בין שידור לצפייה
  - אובייקטים נחוצים: רכיבי ממשק משתמש

מבנה / ארכיטקטורה

הפרויקט מכיל שלושה רכיבים עיקריים: שרת (Server), לקוח משדר (TC) ולקוח מקבל (RC).

לכל אחד מהרכיבים יש חלק משמעותי ושונה בפרויקט

<u>תיאור הארכיטקטורה המוצעת:</u>

<u>שרת (Server):</u>

השרת אחראי על העברת המידע בין הלקוח המשדר (TC) ללקוח המקבל (RC).

נוסף על כך השרת אחראי על פעולת התמלול והכתיבה למסד הנתונים.

השרת מתחלק לשלושה חלקים: TC_Handler, RC_Handler & Transcriptor.

כל אחד מהחלקים אחראי על אחת מהפעולות הראשיות של השרת (קבלה מלקוח משדר, תמלול וסכרון הוידאו והטקסט, העברה ללקוח המקבל).

<u>TC_Handler:</u>

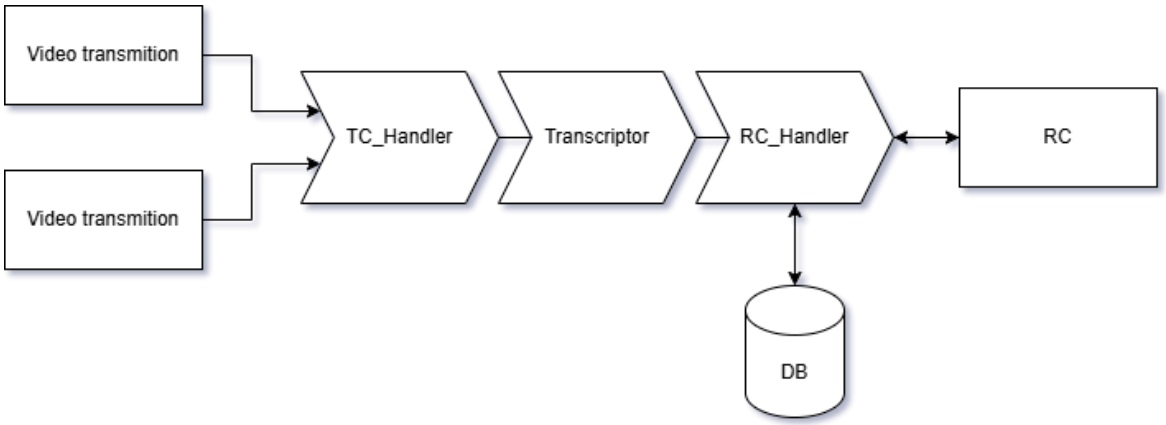
חלק זה בשרת אחראי על קבלת וידאו ואודיו מהלקוח המקבל בעזרת חיבור UDP עם פרוטוקול RTP ומסנכרן ביניהם לפני שהוא מעביר לרכיב השרת הבא.

<u>Transcriptor:</u>

חלק זה לוקח את האודיו ובעזרת קריאת API במטרה לתמלל את האודיו אשר התקבל מה-TC

:RC_Handler

חלק זה אחראי על שליחת הוידאו והתמלול ל-RC ותקשורת עם ממסד הנתונים בעזרת חיבור TCP, HTTP, RTP ו-WebSockets.



לקוח משדר (TC):

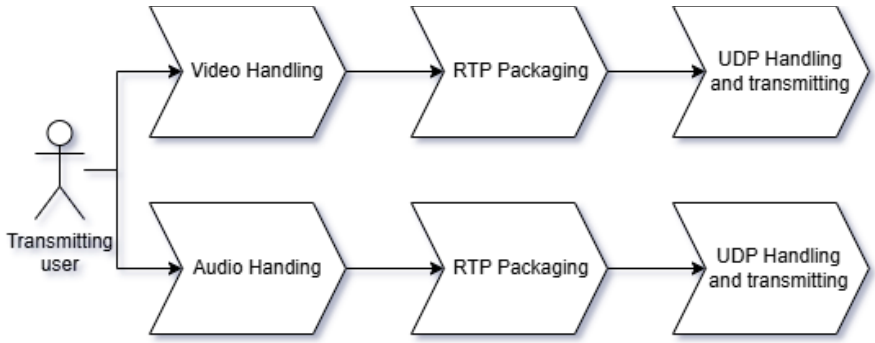
הלקוח המשדר אחראי על שידור הוידאו והאודיו לשרת.

רכיב זה הוא תוכנה לוקלית שמשדרת בעזרת חיבור UDP ופרוטוקול ה-RTP אודיו ווידאו בשני תהליכים נפרדים (פורט לוידאו ופורט לאודיו)

רכיב זה מכיל שני חלקים: Audio_Handler, Video_Handler.

כל חלק מהרכיב בנוי משלושה שלבים עיקריים: קליטה - הכנסה לפקטה – שליחה.

לדוגמא: קליטה של וידאו -> הכנסה לפקטת RTP -> שליחה לשרת.



לקוח מקבל (RC):

תפקידו של רכיב ה-RC הוא הצגת שידורי וידאו ותמלול המתקבלים באופן שוטף מהשרת.

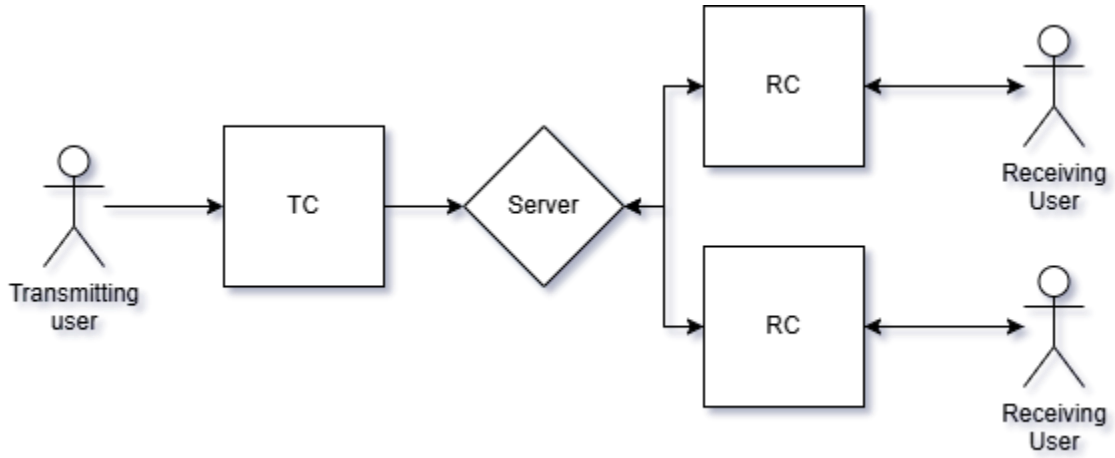
יישום זה מבוסס ווב (Web-based) ומנצל את פרוטוקול WebSocket להעברת נתונים בצורה מהירה ויעילה, המבטיחה תקשורת בזמן אמת (RTC) חיונית לשידורים חיים.

היתרונות בבחירת ארכיטקטורת ווב עבור ה-RC:

- נגישות גבוהה: המערכת נגישה מכל מכשיר המצויד בדפדפן אינטרנט.
- פיתוח ופריסה יעילים: קלות המימוש והתחזוקה בסביבת ווב.
- מתאים למגוון רחב של פלטפורמות (cross-platform): עבודה חלקה על פני מערכות הפעלה ודפדפנים שונים ללא צורך בהתאמות ספציפיות.
- חוויית משתמש נוחה: ממשק מוכר ואינטואיטיבי המקל על השימוש.

מאחר ומדובר באתר כל הקוד של ה-RC מאוחסן בתוך ה-Server אך ניגשים אל ה-RC דרך הדפדפן.

כל שלושת הרכיבים הללו ביחד מנהלים את המוצר הסופי והכרחיים לתפעולו היעיל.



תיאור הטכנולוגיה הרלוונטית

שפות תכנות: השפות השונות המשומשות בפרויקט הזה.

שם השפה	שימוש	סיבה לשימוש
Python	Server, RC, TC	שפה ראשית של הפרויקט. נוחה לשימוש, כתיבה מהירה, מכילה ספריות רבות ומאוד ורסטילית. מוכרת במימוש הנוח שלה לרשתות
TypeScript	RC	משומשת למימוש ה-RC. השפה משומשת בצורה נרחבת בעולם פיתוח הווב. נותנת את כל מה שיש ב-JS אבל עם בטחון של OOP ושימוש ב-Types.
HTML5 & CSS	RC	שפות ידועות לפיתוח ועיצוב אתרים.
SQL	Server	שפה ידועה ובטוחה לכתיבת מסדי נתונים.

מערכות הפעלה: מערכות ההפעלה המשומשות בפרויקט

שם המערכת	שימוש	סיבה לשימוש
Windows	Server, RC, TC	מערכת הפעלה ידועה, נוחה לשימוש, ורסטילית ושומשה לפיתוח ובדיקות.

ניתן להריץ את הפרויקט על כל מערכת הפעלה כל עוד מותקנים התוספים הרלוונטים

פרוטוקולים: הפרוטוקולים המשומשים בפרויקט

שם הפרוטוקול	שימוש	סיבה לשימוש
RTP	Server, RC, TC	פרוטוקול ידוע בתעשייה, נוח לשימוש וורסטילי
HTTP	Server, RC	פרוטוקול בו משתמשים בשביל לתקשר עם ה-browser
Websocket	Server, RC	פרוטוקול מעל HTTP שנותן לשרת יכולת לשלוח לקוח ווב הודעות ללא בקשה של הלקוח.
TCP	Server, RC	פרוטוקול אמין, מוודא הגעת הודעות, חובה לשימוש ב-HTTP
UDP	Server, TC	פרוטוקול מהיר יותר וידוע בשימוש בשביל סטרימינג

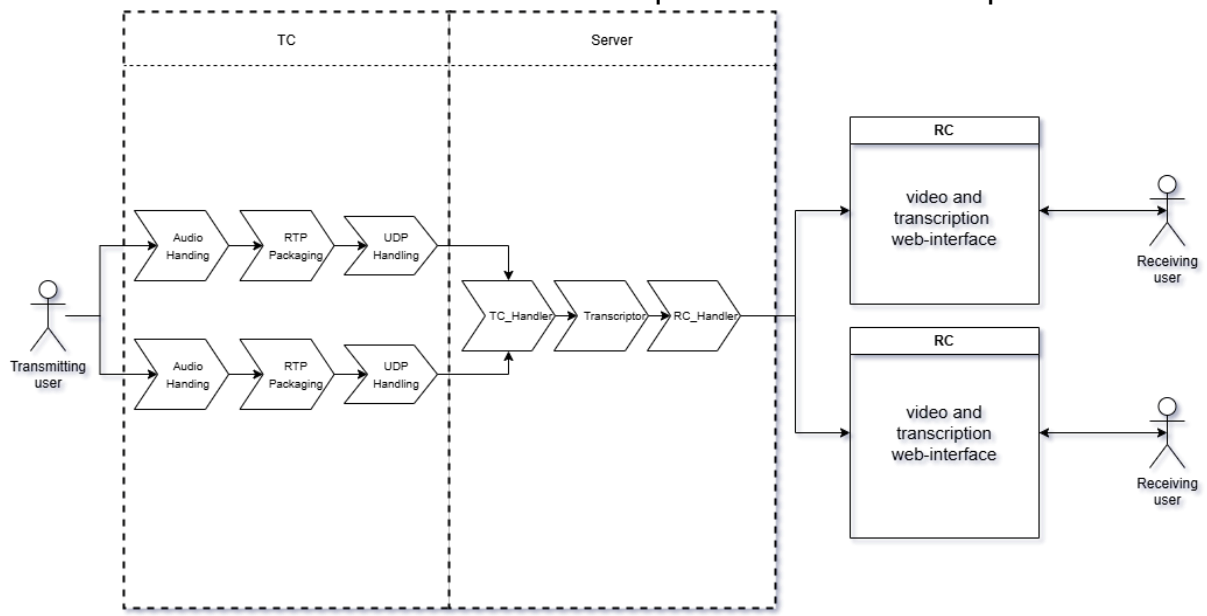
תחומי עניין רלוונטים:

תקשורת, פרוטוקולים, בינה מלאכותית וטכנולוגיות web.

בינה מלאכותית: Vosk מודל תמלול קיים אשר מכיל תמיכה ישירה בפייתון ועבודה offline.

תיאור זרימת המידע במערכת

להלן תרשים המייצג את התהליך של המערכת



תיאור האלגוריתמים המרכזיים

חלוקת פריימים לפקטות והשלמתם בקבלה חזרה לפריימים:

לאחר קבלת הפריים ראיתי שהיו פריימים שהיו בגודל שהיה שדרש יותר מפקטה אחת.
לכן אני החלטתי לבנות אלגוריתם לחלוקת הפריימים לפקטות.

שיטות רעיונות לאלגוריתם זה:

הגדרת מושגים:

- Marker Bit – ביט בפרוטוקול ה-RTP שאפשר להדליק בשביל לסמן בדרים לשרת/לקוח
- Extension – חלק מפרוטוקול הוא ה-extension אשר נותן להוסיף הרחבות לפרוטוקול במידת הצורך.

פתרון #1 – לא נבחר:

לשלוח את הפריים בחלקים, להשאיר לנמען להניח שכל סדרת פקטות שמגיעה בפרק זמן קצר שייכת לאותו פריים.
פתרון זה עלול לגרום לבעיות כאשר יש איבוד פקטות או חפיפה בין פריימים. בנוסף, ניתן להשתמש בגודל פקטות קבוע מראש ללא התאמה לתוכן, מה שיכול לגרום לבזבוז רוחב פס ולפיצול לא אופטימלי.

פתרון #2 –נבחר:

לחלק את הפריימים לכמות פקטות אשר נרשמת בתוך הפקטות של הפריימים בעזרת הוספה של extension לפרוטוקול ה-RTP.
ה-extension יכיל את כמות הפקטות אשר אליהן חולק הפריים.
בפקטה האחרונה אשר תשלח מדליקים את ה-Marker Bit בפרוטוקול.

לאחר קבלת פקטת הmarker בודק השרת אם הוא קיבל את כל הפקטות של הפריים. אם לא הוא ממשיך לחכות אלא אם כן מתחיל להגיע פריים חדש.
במידה ומתחיל להגיע פריים חדש אז השרת זורק את הפריים החסר ומתחיל בקבלה של הפריים החדש.

אני בחרתי בפתרון השני מכיוון שאני הרגשתי שהוא גם הרבה יותר בטוח ויותר מתאים למבנה הפרוטוקול.

תזמון הוידאו והאודיו:

האמת שפה אין כל כך על מה לפרט מבחינת אפשרויות כי לא היו כל כך. יש אחת שאפשר לחשוב עליה שהיא הגיונית והיא לסנכרן את הוידאו והאודיו בעזרת ה-timestamp. ולכן זה הפתרון הנבחר.

הוידאו והאודיו של הלקוח המשדר יחלקו את אותו ה-timestamp בהתחלת השידור ובכך השרת יוכל לתזמן ביניהם

תיאור סביבת הפיתוח

סביבות פיתוח (IDE):

במהלך פיתוח פרויקט זה היה שימוש בשתי סביבות פיתוח:

- Pycharm
- Visual Studio Code

כל סביבה שומשה למטרה ספציפית בה היא מוכרת כהכי נוחה ומכילה כלים העוזרים בפיתוח.

סביבת הפיתוח	שימוש	סיבת הבחירה
Pycharm	פיתוח כל החלקים בפרויקט הדורשים הכתובים ב-Python (TC & Server)	Pycharm היא סביבת הפיתוח הכי ידועה בשביל Python ומכילה כלים רבים (כגון דיבאגר מצוין) ואינטגרציות רבות (כגון גיט) נוסף על כך אני משתמש ב-Pycharm שנים רבות ויודע איך לתפעל את התוכנה.
Visual Studio Code	פיתוח ה-RC.	VSC היא תוכנת עריכת קוד אשר משומשת רבות לפיתוח של אתרים וכתיבת תוכנה לווב. היא מכילה אינטגרציות של צפייה באתר ב"לייב" בנוסף לעוד כל מיני פיצ'רים נוספים העוזרים בבניית אפליקציות ווב התוכנה הזאת היא אחת התוכנות הראשונות שאני כתבתי בהן קוד ואני מכיר אותה היטב, דבר שעזר לי לבחור אותה בשביל הפרויקט הזה
ipython	בדיקות וניסיונות ע פייתון	טרמינל פייתון נוח וזמין שנותן פתרון פשוט ומהיר לעשייה של סניפטים של קוד ובדיקות אבל נותן יותר גמישות מ-Python Idle

מערכת ניהול גרסאות:

בפרויקט זה כמו בכל פרויקט שלי בין אם הוא פרויקט תוכנתי או לא אני השתמשתי ב-git בשביל לשמור את השינויים שלי ולוודא שלא יקרה מצב שמשהו מפסיק לעבוד בפרויקט ואני לא יכול לחזור אחורה.

נוסף על כך על מנת שימוש נוח בגיט וצפייה נוחה בשינויים השתמשתי ב-github התור פלטפורמת ה-git שלי מאחר וזאת הפלטפורמה שאני מכיר הכי טוב.

כלים דרושים לבדיקות:

מאחר ופרויקט זה הוא פרויקט רשתי הכולל את אלמנטים של ווב, פייתון, רשתות, מדיה ועוד כל מיני נדרשים כל מיני כלים אשר עוזרים בבדיקות ודיבאגינג של הקוד והתוצר.

כלי	שימוש	סיבת בחירה
Wireshark	אבחון תקשורת	כלי מוכר, יש לי ניסיון רב איתו ואני מכיר אותו היטב.
Pycharm Debugger	דיבוג סקרפיטים של פייתון	כלי ברמה מאוד גבוהה שנותן תמיכה לדיבוג של תהליכים, תהליכונים רשתות ועוד
Chrome Dev Tools	דיבוג ואבחון בדפדפן	הכלי מובנה בתוך הדפדפן ונותן גישה מלאה לתקשורת גם מוצפנת, קוד FE ונותן לערוך שינויים זמניים בשביל בדיקות
Python logging	שמירה של נתונים בזמן הריצה שאפשר לנתח אחרי	ספרייה בנויה בפייתון שמספקת את הנדרש בצורה יעילה.

תיאור הפרוטוקולים

בפרויקט יש שימוש בשלושה פרוטוקולים – HTTP, RTP, WebSocket

פרוטוקול ה-HTTP (גרסא מצומצמת):

Hypertext Transfer Protocol (HTTP) הוא פרוטוקול תקשורת סטנדרטי להעברת מידע בין לקוחות לשרתים ברשת האינטרנט.

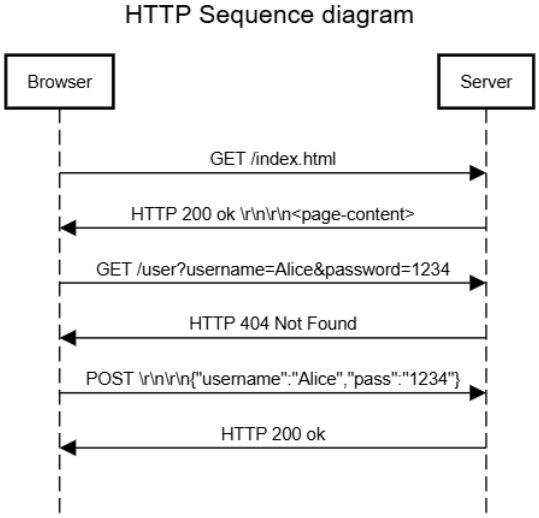
הפרוטוקול פועל במודל בקשה-תגובה, שבו הלקוח שולח בקשה לשרת, והשרת מחזיר את המידע המבוקש.

שיטות HTTP עיקריות:

- **GET** - מיועדת לשליפת נתונים מהשרת. המידע נשלח בתוך כתובת ה URL- ואינו משפיע על נתוני השרת.
- **POST** - משמשת לשליחת נתונים לשרת, כגון נתוני טפסים או קובצי מדיה. הנתונים נשלחים בגוף הבקשה, מה שמאפשר אבטחה גבוהה יותר.

הפרדה בין נתונים בבקשות HTTP:

- (?) – מפריד בין נתיבים למשתנים בבקשות GET, לדוגמא: (example.com/page?user=123)
- (&) - מפריד בין שני משתנים שונים (user=123&age=25)
- (=) - מפריד בין שם המשתנה לערכו (name=John).
- \r\n - מסמן מעבר שורה ומפריד בין כותרות הבקשה (Headers)
- \r\n\r\n – הפרדה בין הכותרות לחלק הגוף של הבקשה בו מעבירים את המ ידע המועבר.



מאפייני HTTP:

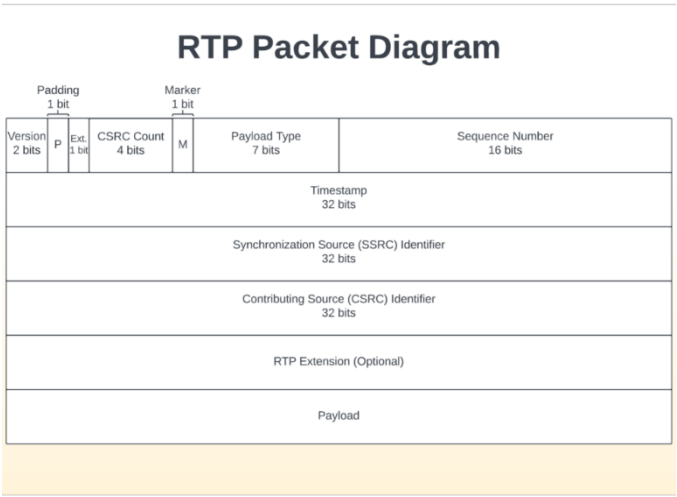
HTTP הוא פרוטוקול stateless, כלומר כל בקשה נשלחת ונענית באופן עצמאי, ללא זיכרון של אינטראקציות קודמות. לשיפור ביצועים ואבטחה.

פרוטוקול ה-RTP:

Real-Time Transport Protocol (RTP) הוא פרוטוקול תקשורת המשמש להעברת מדיה בזמן אמת כגון אודיו ווידאו ברשתות IP. הפרוטוקול פותח כדי לאפשר סנכרון, ניהול רצף נתונים, וזיהוי אובדן פקטות, והוא משמש בעיקר ביישומי סטרימינג ושיחות וידאו.

מבנה פרוטוקול ה-RTP:

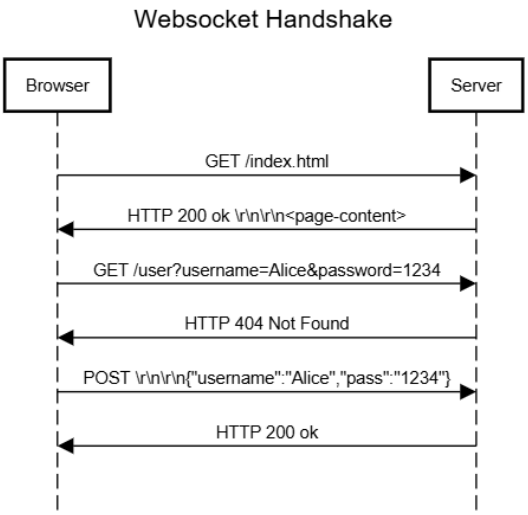
- **Sequence Number** - מזהה את סדר הפקטות כדי לאפשר שחזור נכון של הנתונים.
- **Timestamp** - מסייע בסנכרון בין זרמי מדיה שונים.
- **Payload Type** - מציין את סוג הנתונים (למשל, קידוד אודיו או וידאו).
- **Synchronization Source (SSRC)** - מזהה את מקור הנתונים כדי להבחין בין זרמים שונים.
- **Contribute Source Identifiers (CSRC)** - מזהה מקורות נוספים של מדיה (למשל, כשכמה משתמשים משתתפים בשיחת ועידה קולית, כל אחד מהם מקבל מזהה ייחודי בתוך הפקטות).
- **Extension Bit (X)** - אם מוגדר ל 1-,זה מציין כי קיימים שדות הרחבה נוספים בפקטה. משתמשים בזה כדי להוסיף מידע נוסף מעבר למבנה הבסיסי של RTP
- **Marker Bit (M)** - מסמן פקטה מיוחדת, למשל פקטה שמסמנת את סוף מסגרת וידאו או נקודת התחלה של מקטע חשוב.
- **Payload Type (PT)** - מזהה את סוג הנתונים (לדוגמה, קידוד אודיו/וידאו כמו H.264 או Opus)



פרוטוקול ה-WebSocket:

פרוטוקול הווב סוקט הוא רוטוקול המאפשר תקשורת רציפה מעל תקשורת אינטרנטית לעומת פרוטוקול ה-HTTP שהוא פרוטוקול stateless.

פרוטוקול ה-Websocket מתחיל בלחיצת יד פשוטה דרך פרוטוקול ה-HTTP כמתואר בתרשים בצד.

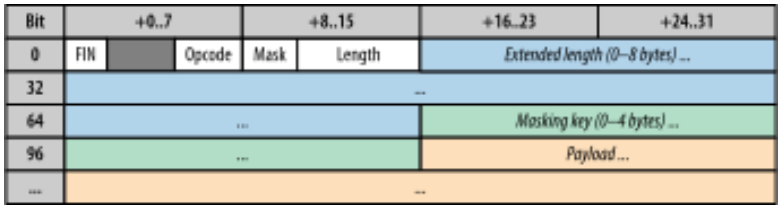


לאחר לחיצת היד הצדדים המחוברים יכולים להתחיל לשלוח הודעות בעזרת נפרמטרים של הפרוטוקול:

- FIN (1 bit) – מסמן אם זאת הפקטה האחרונה בהודעה.
- RSV1..3 (1 bit each) – ביטים שמורים להרחבות של הפרוטוקול
- Opcode (4 bits) – סוג התוכן (בינארי, טקסט וכ"ו)
- MASK (1 bit) – מציין אם יש מפתח הצפנה בשימוש
- Payload Length (7 bit) – גודל הנתונים המועברים בפקטה. חלק זה יותר מסובך ובנוי בצורה הבאה. במידה ויש צורך בהרחבה אז מתווסף באפר שנקרא (Extended payload length) שהוא תוספת של בתים לאורך הנתונים

גודל נתונים	Payload Length גודל	Payload Length ערך
עד 125 בתים	7 ביט	גודל הנתונים
125 - 65,535 בתים	23 ביט (עוד 2 בתים)	7 ביט התחלתי = 126 שני בתים נוספים = גודל הנתונים
65,535 ומעלה	87 ביט (8 בתים נוספים)	7 ביט התחלתי – 127 8 בתים נוספים = גודל הנתונים

- Masking Key (4 bytes) – מפתח אקראי המשמש להצפנת הנתונים, **נדרש רק בהודעות מהלקוח לשרת.** לא קיים בהודעות מהשרת ללקוח.
- Payload Data – הנתונים עצמם אשר מועברים בפקטה



תיאור מסכי המערכת

מסכי ה-RC:

מסך פתיחה:

בפתיחת המסך מוצגות שתי אפשרויות – Connect & Conversations

Join stream – התחברות לשידור

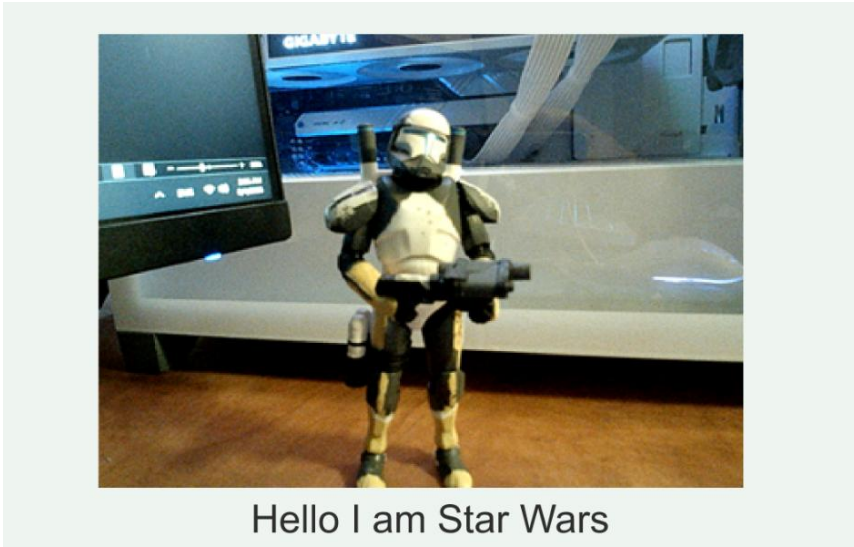
View saved – צפייה בשיחות אשר תועדו.

עיצוב:

מסך שידור:

במסך מוצג השידור המתומלל הנשלח מהשרת. במידה ואין שידור תוצג הודעה

עיצוב:



מסך שיחות שתועדו:

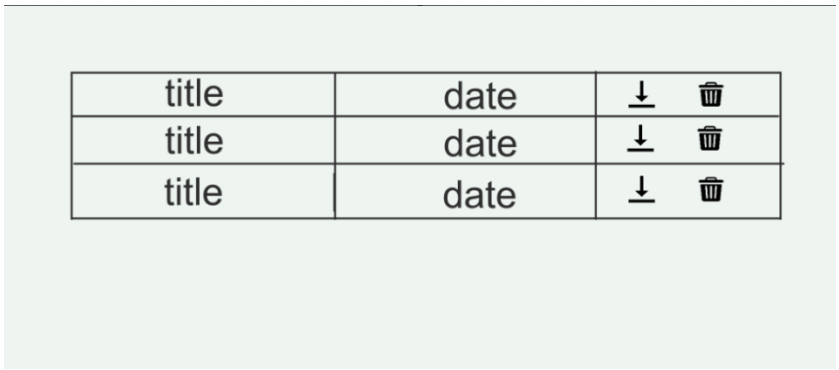
במסך מוצגת רשימה של כל השיחות שתועדו – כותרת, ותאריך

לכל שיחה יש שתי אפשרויות, הורדה ומחיקה.

הורדה: הורדה של תמלול השיחה בקובץ טקסט לטלפון

מחיקה: מחיקת תיעוד השיחה מממסד הנתונים.

עיצוב:

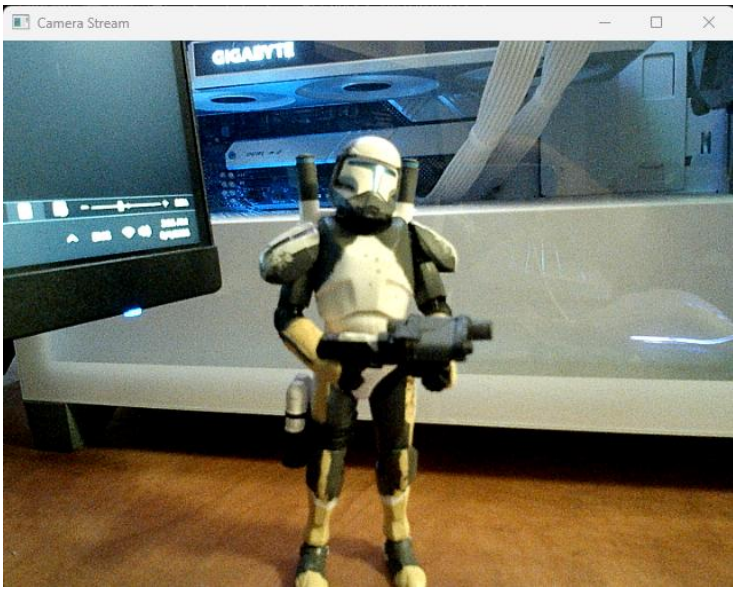


מסכי ה-TC:

מסך שידור:

מסך המציג את המצלמה והשידור

עיצוב:



תיאור מבני הנתונים

הפרויקט מכיל ממסד נתונים אחד ופשוט ממסד נתונים של השיחות השונות אשר תועדו וככה הוא נראה:

Id: int	Title: string	Date: string	Filename: string
1	Conversation-1	23/5/2024	Conversation-1-23/5/2024.txt

בטבלה מוצגות כל השיחות השמורות ע"פ מספר מזהה ייחודי, הכותרת, התאריך ושם הקובץ של השיחה.

סקירת חולשות בפרויקט

פרויקט זה אינו מאוד מסובך לתפעול מבחינת אבטחה מאחר שאינו מכיל הרבה דברים.

ממסד נתונים:

כדי להגן מפני פריצות SQL Injection יש שימוש בשאילתות פרמטריות שהן חלק מספריית SQLite בשביל להגן מפני המתקפות הללו. השאילתות בנויות מיישום של (?) בכל מקום בו נועד להכנס מידע ולאחר מכן העברת המידע לפי סדר שלאחר מכן הספרייה בודקת ומאשרת.

Web:

מבחינת הווב הכל משתמש בהצפנת SSL/TLS (Transport Layer Security) ותעודה חתומה עצמאית (ssc) בשביל לשמור על כל המידע מוצפן ומאובטח. בשביל הגנה ממתקפות MITM.

UDP Connection:

בחלק זה גם כן יש שימוש בהצפנת DTLS (Datagram Transport Layer Security). הצפנה זאת חוסמת MITM ומאזינים לא רצויים.

מימוש הפרויקט

ספריות ומודולים ששומשו בפרויקט

שם המודול / ספרייה	סיבת השימוש	סיבת הבחירה
socket	ניהול תקשורת	ספרייה שמובנת בתוך Python, נוחה לשימוש, אני מכיר אותה היטב וידועה לניהול תקשורת.
os	אינטראקציה עם מערכת ההפעלה	מאפשר גישה לשירותי מערכת ההפעלה שמשומשים בפרויקט.
logging	ניהול ורישום לוגים	כלי מובנה לניהול לוגים בצורה מסודרת, גמיש וניתן להתאמה אישית בקלות.
time	ניהול זמן ושעון (fps)	מספק פונקציות לעבודה עם זמן, כולל מדידת ביצועים, השהיות, וקבלת תאריך נוכחי.
select	ניהול לקוחות מרובים ב-RC	מאפשר ניהול לקוחות מרובים בשרתים בצורה פשוטה ונוחה
re	עיבוד ביטויים רגולריים	משמש לחיפוש, התאמה ומניפולציה של טקסטים עם ביטויים רגולריים, תומך בגמישות מירבית.
struct	עבודה עם מבני נתונים בינאריים	מאפשר המרה של נתונים בין פורמט בינארי לפורמט ש-Python מבין, חיוני בתקשורת והמרת נתונים.
zlib	דחיסת נתונים של וידאו ואודיו ב-TC	נוחה ופשוטה לשימוש ובנויה בתוך python

numpy	עיבוד וידאו	ספרייה נוחה ומוכרת שנותנת תמיכה ב-opencv.
cv2	משומשת לקליטת וידאו מה-TC ולקריאת הוידאו ב-Server	מוכרת בעולם, נוחה לשימוש ויש לי ניסיון רב איתה מפרויקטים קודמים

שם המודול / ספרייה	סיבת השימוש	סיבת הבחירה
pyaudio	קליטת אודיו מה-TC	מאפשר ניהול והקלטה של אודיו, מתאים לעיבוד קובצי קול ולזיהוי דיבור.
math	פונקציות מתמטיות מתקדמות	מובנה בתוך python ונוח לשימוש.
random	יצירת מספרים רנדומליים ssrc (מספר מזהה בפרוטוקול RTP)	מובנה בתוך python, נוח וקל לשימוש.
threading	ניהול Threads ב-Server	מאפשר יצירה ועובדה של תהליכים בתוך התוכנה.
multiprocessing	ניהול תהליכים ב-TC שליחת אודיו ושליחת וידאו	מספק תמיכה לריבוי תהליכים, משפר ביצועים במערכות עם עיבוד כבד ע"י הפעלת מספר ליבות של המעבד.
ssl	http and websocket encryption	ספרייה שהדפדפן יודע לעבוד עם המידע שהיא שולחת אליו ומטפלת בכל העניין אוטומטית.
dtls	ספרייה משומשת להצפנת זרמי UDP	לפי מחקר אינטרנטי הספרייה הכי מומלצת להצפנת UDP
sqlite3	ניהול DB	ספרייה שאני משתמש כבר הרבה שנים ומכילה פרמטריזציה מובנת
vosk	תמלול	נותן אפשרות לשימוש אופליין של מודלים לוקליים לתמלול טקסט

מודולים ומחלקות שאני פיתחתי

AudioReceiveHandler

שימוש: Handles real-time UDP audio reception with delta time synchronization.

תכונות: אין

Method name	Docs
recv_audio	Receives RTP audio packets and plays audio in real-time using delta time for better timing. :return: None

VideoTransmission

שימוש: Bundles video handlers and handles transmission

תכונות:

- video_capture_source – camera port
- port – port of udp connection
- payload_type – rtp payload type of video
- rtp_handle – instance of RTPHandler
- udp_handle – instance of UDPClientHandler
- logger – class logger

Method name	Docs
transmit_video	transmit video using rtp :return: None

VideoCapture

שימוש: This class handles the video capture and handling.

תכונות:

- logger – class logger
- source – camera source port
- capture – cv2 capture instance

Method name	Docs
get_frame	Retrieves a single frame from the video source. :return tuple: (success, frame), where `success` is a boolean indicating if the frame was read successfully, and `frame` is the captured frame.
release	Releases the video capture resource. :return: None
__del__	Ensures resources are released when the instance is deleted. :return: None

UDPClientHandler

שימוש:

This class represents the video transmitting client using UDP.
It initializes an RTPHandler instance to handle RTP packet creation
and uses a UDP socket for sending packets.

תכונות:

- server_address – server host
- server_port – udp binding port
- sock – udp socket
- logger – class logger

Method name	Docs
send_packets	Captures video frames, creates RTP packets, and transmits them to the server via UDP. :param packets: (list[bytes]) packets to send :return bool: whether operation was successful

RTPHandler

שימוש:

This class handles the RTP protocol, including creating and parsing RTP packets.

תכונות:

- payload_type – rtp payload type
- ssrc – rtp ssrc
- sesquence_number – rtp sequence number (init 0)
- logger – class logger

Method name	Docs
build_header	Constructs the RTP header. :param extension_data: :param marker: (int) The marker bit. :param csrcs: (list[int]) List of contributing source identifiers. :return: bytes: The RTP header as a byte string.
create_packets	Creates an RTP packet by combining the header and payload. :param payload: (bytes) payload to put in the rtp packet :param csrcs: (list[int]) List of contributing source identifiers. :return: bytes: The complete RTP packet.
get_ssrc	Returns the SSRC for the RTP stream. :return: int: The SSRC value.
_update_timestamp	updates timestamp :return: None

AudioTransmissionHandler

שימוש:

Handles real-time audio transmission via RTP over UDP with delta time synchronization.

תכונות:

- logger – class logger
- audio_capture – pyaudio audio capture instance
- rtp_handler – rtpHandler instance
- udp_handler – udpClientHandler instance

Method name	Docs
start_streaming	Begins live audio transmission with delta time control for better synchronization. :return: None
stop_streaming	Cleans up resources after transmission stops. :return: None

AudioCapture

שימוש:

Handles real-time audio capture and streaming

תכונות:

- rate – sample rate (const)
- channels – channels (const)
- chunk_size – hunk size (const)
- logger -class logger
- audio – PyAudio instance
- stream – opening input stream in pyaudio

Method name	Docs
get_audio_chunk	Captures and returns a single chunk of audio data. :return bytes: Raw audio data.
release	Cleans up resources. :return: None

VideoReceiveHandler

שימוש:

Handles real-time UDP video reception

תכונות: אין

Method name	Docs
recv_video	Receives video and presents it :return: None

UDPServerHandler

שימוש:

This class represents the video receiving server using UDP.
It initializes a UDP socket to listen for incoming packets and uses RTPacketDecoder to decode the received RTP packets.

תכונות:

- bind_address – (const) the server address
- bind_port – port to listen on
- sock – udp socket
- _uncompleted_frame_packets – dictionary used to assemble messages
- logger – class logger

Method name	Docs
_receive_packet	Will receive a packet using a UDP connection :param buffer: (int) the buffer to receive :return bytes | None: The received packet if one was received
assemble_rtp_message	Assembles / Discards frames :param seq_start: (int) fragment sequence start :param seq_end: (int) fragment sequence end :return bytes | None: bytes if a frame was assembled, None if frame was dropped
receive_rtp_message	Receives RTP packets from the client & decodes them. :return bytes or None: If a message/packet is received then the payload will be returned.

RTPPacketDecoder

שימוש:

Decodes RTP packets into header fields and payload.

תכונות:

- packet – the raw packet
- version – rtp version
- padding – rtp padding bit
- extension – rtp extension bit
- csrc_count – rtp csrc count
- marker – rtp marker bit
- payload_type – rtp payload type
- sequence_number – rtp sequence number
- timestamp – rtp timestamp
- ssrc – rtp ssrc
- csrcs – rtp csrcs list
- extension_data – holds the data of the extension if there is one
- payload – rtp payload
- logger – class logger

Method name	Docs
decode_packet	Decodes the RTP packet, extracting its header and payload. :return: None
get_header_info	Returns the RTP header fields as a dictionary.

	:return: dict: Header fields with their values.
get_payload	Returns the payload from the RTP packet. :return: bytes: The raw payload data.

Transcriptor

מימוש:

Handles live speech recognition and captioning using AudioCapture.

תכונות:

- model – vosk model instance
- recognizer – vosk voice recognizer
- logger – class logger
- audio_capture – audioCapture instance

Method	Docs
transcribe_audio	Continuously captures and transcribes live audio. :return: None

HttpParser

מימוש:

Class to make http requests usable in code more easily

תכונות:

- METHOD – http method (get / post)
- URI – uri of the request
- HTTP_VERSION – version from the request
- QUERY_PARAMS – query parameters given in the request
- HEADERS – headers given in the request
- BODY – body of the request
- COOKIES – cookies given in the request

Method	Docs
__header_parser	Extracts the headers from the request :return dict[bytes]: {header: value}
__body_parser	Extracts the body from the requests :return bytes: the body of the request
__section_one_parser	Splits the first section of the http_ophir request (before the headers) :return tuple: (method, uri, http_version, query_params)
__str__	Str dunder function for the HttpMsg class :return str: The http_ophir message in full

HttpMsg

מימוש:

Create easy to use http messages including responses and requests

תכונות:

- error_codes – error codes in the http message
- body – body of the http message
- headers – dict of the headers

Method	Docs
__error_code_finder	Returns the error code and message using the error code. :param error_code: The error code to be found :return bytes: The error code
prettify	Prints the message in a readable detailed format :return: None
__build_headers_bytes	Formats the headers into bytes :return bytes: Byte string with the headers formatte
build_message_bytes	Builds the http_ophir message :return bytes: The http_ophir message
__str__	Str dunder function for the HttpMsg class :return str: The http_ophir message in full

App

מימוש:

The Base for the web app

תכונות:

- routes – dict holding all the routes
- four_o_four – holds the path to the default 404 page
- __closed – bool is connection closed

Method	Docs
route	Route decorator - adds the route to the routes' dictionary. :param permission_cookie: The cookie name that allows permission to page. :param route: the route for the uri :return: what the original function needs to return
__receive_message	Receives a message from a client :param client_socket: :return bool or HttpParser: False if the message is invalid, HttpParser with the message
__handle_client	Handles the client input. :param request: The request that the server got. :param client_socket: The client socket. :return: None
set_four_o_four	Sets the route to the 404 page. :param route: The route to the 404 page html. :return: None

close_app	Shuts down the server and closes the server :return: None
run	Starts the http server. :return: None

מסמך בדיקות מלא

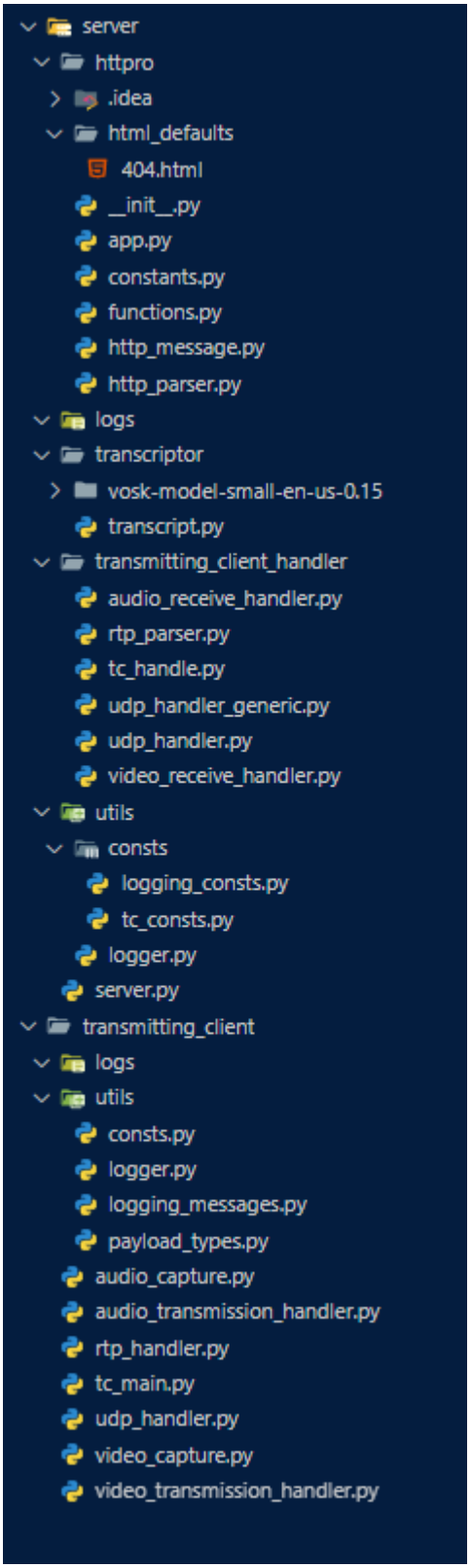
שם הבדיקה (שיעיד על התוכן)	מה אמורה לבדוק	דרך הבדיקה והאם צלחה	דרך התמודדות
בדיקת חיבוריות - חיבור למצלמות אלחוטיות	חיבור מהיר ויציב למצלמה המוגדרת	בדיקה של חיבור למצלמה המוגדרת. דיווח על שגיאות חיבור (אם קיימות). נכשל	בדיקה של פרוטוקול ה-RTP ותיקון מימושו
בדיקת חיבוריות - חיבור למיקרופון	זיהוי וחיבור תקין למיקרופון המוגדר	בדיקה של חיבור למיקרופון המוגדר. דיווח על שגיאות חיבור (אם קיימות). עבר בהצלחה	
בדיקת חיבוריות - חיבור לשרת	חיבור מהיר ויציב לשרת התמלול	בדיקה של חיבור לשרת. דיווח על שגיאות חיבור (אם קיימות). עבר בהצלחה	
בדיקה פונקציונלית - לכידת וידאו	לכידה תקינה של וידאו מכל המצלמה המחוברת	בדיקה של לכידת וידאו מכל המצלמות. וידוא איכות וידאו משתנה (רזולוציה, קצב פריימים). עבר בהצלחה	
בדיקה פונקציונלית - קליטת אודיו	קליטה תקינה של אודיו מהמיקרופון המחובר	בדיקה של קליטת אודיו מהמיקרופון. וידוא איכות שמע משתנה (עוצמה, רעשי רקע). עבר בהצלחה	

בדיקה פונקציונלית - תמלול בזמן אמת	תמלול אודיו בזמן אמת בצורה מדויקת	שימוש בקובץ אודיו מקומי לבדיקה בדיקה אוטומטית של התמלול המתקבל. וידוא תמיכה בשפות שונות, טיפול בהפסקות בדיבור. נכשל	בדיקה על הליך התמלול ובדיקת אלטרנטיבות

בדיקה פונקציונלית - הצגת כתוביות	הצגה תקינה של כתוביות על המסך	בדיקה של הצגת כתוביות על המסך. וידוא סנכרון בין כתוביות לווידאו, אפשרויות עיצוב כתוביות. נכשל	שכתוב פעולת הצגת הכתוביות בזמן אמת
בדיקת משאבי מערכת	צריכת משאבים (CPU, זיכרון) תקינה בעת הפעלת התוכנה	ניטור של צריכת משאבים. דיווח על חריגות. עבר בהצלחה	

מדריך למשתמש

עץ קבצים



התקנת TC והרצתו

בשביל להתקין את ה-TC יש להתקין את פייתון דרך האתר הרשמי של פייתון ולהריץ לאחר מכן את הפקודות הבאות:

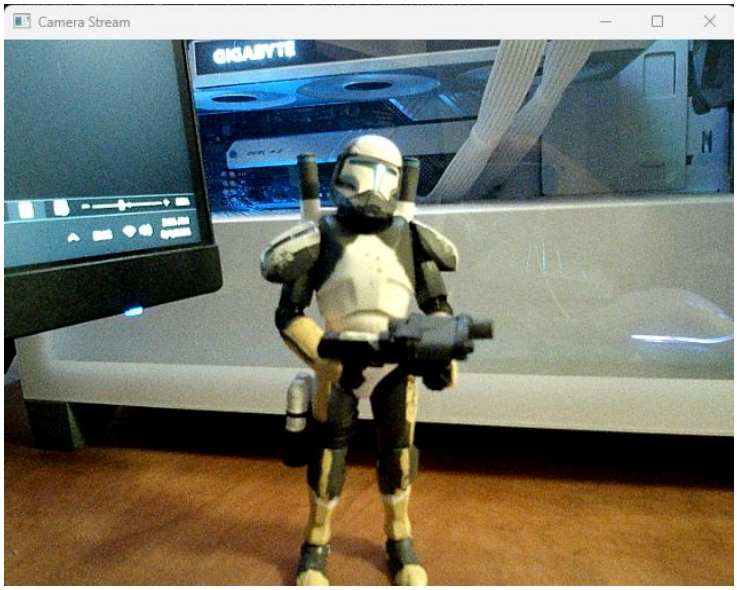
1. pip install opencv
2. pip install numpy
3. pip install pyaudio
4. pip install dtls

לאחר התקנת הספריות יש להתקין את תוכנת ה-TC.

עכשיו הכל מוכן ואפשר להריץ את הקובץ הראשי אשר נקרא "tc_main.py" דרך ה-cmd אשר הוכוון לתיקייה אשר בה התקנתם את התוכנה. לדוגמא:

```
D:\Python\ACE>python3 tc_main.py
```

לאחר שיעלה חלון המצלמה יתחיל השידור.



התקנה והרצת השרת וה-RC

בשביל להתקין את ה-TC יש להתקין את פייתון דרך האתר הרשמי של פייתון ולהריץ לאחר מכן את הפקודות הבאות:

1. pip install opencv
2. pip install numpy
3. pip install pyaudio
4. pip install dtls
5. pip install vosk

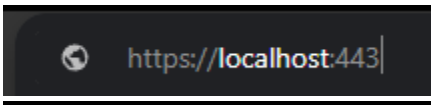
עכשיו הכל מוכן ואפשר להריץ את הקובץ הראשי אשר נקרא "server.py" דרך ה-cmd אשר הוכוון לתיקייה אשר בה התקנתם את התוכנה. לדוגמא:

```
D:\Python\ACE>python3 server.py
```

עכשיו השרת רץ ואפשר להתחיל להתחבר ל-RC.

התחברות ל-RC דרך השרת הלוקלי מתבצעת בצורה הבאה:

1. הכנסו לדפדפן שלכם והקלידו את הדבר הבא בשורת החיפוש:



2. תבחרו אם אתם מעוניינים לראות את השיחות אשר היו או להצטרף לשידור

במידה והצטרפתם לשידור השידור יוצג במסך הבא:

Hello I am Star Wars

במידה ובחרתם לראות את השיחות יוצג המסך הבא: בו מוצגות האפשרויות להוריד את התמלול או למחוק אותו

title	date	⭳ 🗑
title	date	⭳ 🗑
title	date	⭳ 🗑

רפלקציה אישית

אני התחלתי את תהליך העבודה על הפרויקט באיזור פברואר אך לצערי אני לקחתי על עצמי יותר מדי.

תוך כדי התמודדות עם הפרויקט אני הייתי צריך להקים רסיטל, לתחזק את העבודה שלי, ולהתמודד עם עוד מיליון מחויבויות שונות שהיו לי והאמת שזה היה יותר מדי.

הצבתי לעצמי לוח זמנים מפורט ופייר לפרויקט אך נכשלתי קשות בלעמוד בו והאתגר הכי גדול שלי היה לגרום לעצמי למצוא זמן לכל הדברים ביחד ובעיקר בהם הפרויקט.

היו פעמים שהצלחתי ורוב הדברים גם לקחו יותר זמן ממה שחשבתי וגם היו יותר מאתגרים (כמו מימוש הפרוטוקולים). בסופו של דבר מועד ההגשה התקרב ואני הבנתי שאני צריך לקחת את עצמי בידיים והתחלתי לעבוד יותר ברצינות והתחלתי לקבוע עם חברים לעבוד ביחד אך לצערי זה היה מאוד מאוחר ולמרות שהספקתי הרבה אני לא הספקתי מספיק.

תמיד היו לי בעיות של לדרום לעצמי להרגיע, לבחור ולשבת ולעשות משהו שמלחיץ אותי אך הפרויקט הזה הראה לי כמה הכישור של שליטה עצמית וניהול זמנים הוא חשוב וקריטי.

מן הפרויקט אני למדתי כל מיני כלים חדשים של דיבוג, ניהול זמנים, ניהול ציפיות, עבודה מסודרת ועוד. אבל הדבר שאני מוצא כהכי משמעותי הוא שלבי התכנון של הפרויקט. אני יודע שיש הרבה שמזלזלים בחשיבות של שלבי הארכיטקטורה או כל שלבי התכנון ופשוט רוצים לצלול ישר לכתיבת הקוד (ואני הייתי ביניהם) אך תכנון הפרויקט הראה לי כמה התכנון הוא דבר חשוב ועוזר למען הפרויקט ולמען איך שהוא ייראה ויעבוד וזהו דבר שאני חד משמעית איישם לפרויקטי המשך ולשירות הצבאי שלי.

מבחינת יישום חלקי הפרויקט הדבר העיקרי שאני הייתי מיישם אחרת הוא התמלול. אני הייתי מטפל בו ישירות בלקוח המשדר ולא בשרת מאחר שזה יותר ישיר ויגרום לפחות בעיות סנכרון ככל הנראה במיוחד אם יתוזמן נכון בעזרת פרוטוקול ה-RTP. מעבר לזה אני מאוד מרוצה מהארכיטקטורה ואיך שהפרויקט בנוי ומאורגן.

אם היו משאבים נוספים אני הייתי מרחיב את הפרויקט לצד החומרתי ובונה מערכת שבאמת אפשר להרכיב על רכבים אמיתיים. נוסף על כך אני הייתי מוסיף אפשרות ליותר ממצלמה אחת כדי שתהיה לכל נוסע מצלמה משלו.

הפרויקט הזה הראה לי הרבה מאוד דברים על עצמי גם כאלה רעים וגם טובים ואני חושב שהוא בעיקר הראה לי נקודות לשיפור בהתנהלות שלי עם אחריות וניהול ציפיות.

ביבליוגרפיה

WebSocket

- **Mumtaz, W. (n.d.). *WebSocket-Video-Processing*. GitHub. Retrieved June 4, 2025, from https://github.com/waqarmumtaz369/WebSocket-Video-Processing**
- **(n.d.). *WebSocket*. High Performance Browser Networking. Retrieved June 4, 2025, from https://hpbn.co/websocket/**

RTP

- **Wikipedia. (n.d.). *Real-time Transport Protocol*. Wikipedia. Retrieved June 4, 2025, from https://he.wikipedia.org/wiki/Real-time_Transport_Protocol**

HTTP

- **Wikipedia. (n.d.). *Hypertext Transfer Protocol*. Wikipedia. Retrieved June 4, 2025, from https://he.wikipedia.org/wiki/Hypertext_Transfer_Protocol**

דברים שעשיתי בעזרת בינה מלאכותית:

Microsoft copilot:

1. מחקר על הצפנות
2. מחקר על RTP
3. מחקר על WEBSOCKET
4. דיונים על באגים
5. התייעצות כללית
6. התייעצות על ספר פרויקט
7. פירוט מודולים וספריות

Gemini:

1. מחקר על HTTP
2. כתיבת ביבליוגרפיה

נספחים

server\httpro__init__.py

```
"""
    AUTHOR: Ophir Nevo Michrowski
    DATE: 15/03/24
    DESCRIPTION: HTTP protocol package. Includes (http parser and formatter, http
request builder)
"""
import logging
import os.path
import httpro.constants as consts
import httpro.http_parser
import httpro.http_message
import httpro.app
import httpro.functions
import httpro.constants


def http_setup() -> None:
    """
    Function that contains all the auto_checks for the http package and sets up
logger
    :return: None
    """
    httpro.http_parser.auto_test_http_parser()
    httpro.http_message.auto_test_http_message()

    # is_valid_request auto tests #
    request: bytes = b"GET /index.html HTTP/1.1\r\nHost: 192.168.37.45\r\n\r\n"
    request_invalid: bytes = b"GET /index.html
HTTP/1.1\r\nHost:192.168.37.45\r\n\r\n"
    assert is_valid_request(request)["valid"], is_valid_request(request)["reason"]
    assert not is_valid_request(request_invalid)["valid"]

    # Start logger #
    if not os.path.isdir(consts.LOG_DIR):
        os.makedirs(consts.LOG_DIR)

    consts.HTTP_LOGGER.setLevel(consts.LOG_LEVEL)
    file_handler = logging.FileHandler(consts.LOG_FILE_NAME)
    file_handler.setFormatter(logging.Formatter(consts.LOG_FORMAT))
    consts.HTTP_LOGGER.addHandler(file_handler)

    consts.HTTP_LOGGER.info("httpro initiated")


def is_valid_request(request: bytes) -> dict[bool, str] or dict[bool]:
    """
```

```
    Checks if a http request is valid.
    :param request: the request to validate.
    :return dict[bool, string]: {"valid": True/False, "reason":"reason"}
    """
    # Making sure that the argument is correct #
    if not isinstance(request, bytes):
        return {"valid": False, "reason": "request data type must be bytes"}

    # Split the request into lines #
    lines = request.decode('utf-8').split('\r\n')

    # Check if there's at least one line (the request line) and a blank line
separating headers and body #
    if len(lines) < 2 or b'\r\n\r\n' not in request:
        return {"valid": False, "reason": "At least one line (the request line) and a
blank line separating headers "
                                            "and body are needed."}

    # Check the request line for the correct number of elements and separators #
    request_line_parts = lines[0].split(' ')
    if len(request_line_parts) < 3:
        return {"valid": False, "reason": "Incorrect number of elements or
separators."}

    # Ensure method, path, and HTTP version are correctly separated #
    method, path, version = request_line_parts[0], request_line_parts[1],
request_line_parts[-1]
    if method.encode() not in consts.REQUEST_TYPESs():
        return {"valid": False, "reason": f"{method} is not a real method in
httpro."}

    # Check if the request has a http version #
    if not method or not path or not version.startswith('HTTP/'):
        return {"valid": False, "reason": "Request must have httpro version."}

    # Check headers for correct formatting
    is_host_header = False
    for header in lines[1:-2]:  # Ignoring the request line and the last two elements
(the last header and the body)
        if ': ' not in header:
            return {"valid": False, "reason": f"{header} header does not contain
colon-space separator."}
        if "Host" in header:
            is_host_header = True  # Mandatory host header is in the request

    # Check if the request has mandatory host header #
    if not is_host_header:
        return {"valid": False, "reason": "Host header is mandatory."}

    # Return true if the request is valid #
    return {"valid": True}
```

```
def read_file(path: str) -> http_message:
    """
    Create a request for a html page.
    :param path: the path to the html.
    :return http_message: the http formatted message containing the html file.
    """
    if not os.path.isfile(path):
        consts.HTTP_LOGGER.Exception("File not found.")
        raise FileNotFoundError("File not found.")
    else:
        with open(path, 'rb') as f:
            return f.read()
```

server\httpro\app.py

```
"""
    AUTHOR: Ophir Nevo Michrowski
    DATE: 15/04/24
    DESCRIPTION: This is the app class. It will handle the webapp and the path.
"""
# Imports #
import socket
import time

import select
import httpro.constants as consts
import logging
import re
import httpro.http_parser
import os

# global vars #
global readable_socks_list, writeable_socks_list, exception_socks_list


class App:
    """The Base for the web app."""

    # ---------- Constructor ---------- #
    def __init__(self):
        """
        Constructor of class
        :return: None
        """
        self.routes = dict()
        self.four_o_four = consts.FOUR_O_FOUR
        self.__closed = False

    # ---------- Decorators ---------- #
    def route(self, route: bytes, permission_cookie: bytes = "None"):
        """
        Route decorator - adds the route to the routes' dictionary.
        :param permission_cookie: The cookie name that allows permission to page.
        :param route: the route for the uri
        :return: what the original function needs to return
        """

        def add_to_route_dict(original_function):
            """
            This is the decorator function
            :param original_function: original function that decorator decorates
            :return: what the original function needs to return
            """
            try:
                self.routes[route] = original_function, permission_cookie
```

```
            except TypeError:
                consts.HTTP_LOGGER.debug("routes is not initialized.")

            def wrapper_function(*args, **kwargs):
                """
                This is the wrapper function.
                """
                raise Exception("This is a route function. It cannot be run.")

            return wrapper_function

        return add_to_route_dict

    # ---------- Private functions ---------- #
    @staticmethod
    def __receive_message(client_socket: socket) -> bool or
httpro.http_parser.HttpParser:
        """
        Receives a message from a client
        :param client_socket:
        :return bool or HttpParser: False if the message is invalid, HttpParser with
the message
        """
        try:
            message = b""

            # Multiple receives to eliminate server blocking without too many
requests with the timeout #
            time_start = time.time()
            while b"\r\n\r\n" not in message and time.time() - time_start <
consts.RECV_TIMEOUT:
                msg = b""
                try:
                    client_socket.settimeout(.5)
                    msg = client_socket.recv(consts.RECV_LENGTH)
                except socket.timeout:
                    consts.HTTP_LOGGER.debug("Got timeout on socket receive")
                else:
                    consts.HTTP_LOGGER.debug("Success on socket receive.")

                if not msg:
                    continue
                message += msg

            try:
                content_length = int(re.search(rb'Content-Length: (\d+)',
message).group(1))
                # Check if body was already received #
                if len(message.split(b'\r\n\r\n')) > 1 and
len(message.split(b'\r\n\r\n')[1]) != content_length:
                    body = b''
                    # Receiving body #
```

```
                    while len(body) < content_length:
                        chunk = client_socket.recv(consts.RECV_LENGTH)
                        if not chunk:
                            consts.HTTP_LOGGER.debug("Ended body receive")
                            break
                        body += chunk

                    message += body

            except AttributeError:
                consts.HTTP_LOGGER.debug("The message does not have Content-Length
header")

            except Exception as e:
                consts.HTTP_LOGGER.exception(e)

            return httpro.http_parser.HttpParser(message)

        except Exception as e:
            consts.HTTP_LOGGER.exception(e)

    def __handle_client(self, request: httpro.http_parser.HttpParser, client_socket:
socket) -> None:
        """
        Handles the client input.
        :param request: The request that the server got.
        :param client_socket: The client socket.
        :return: None
        """
        response: httpro.http_message

        if request is not None and request.URI is not None:
            if request.URI in self.routes.keys() and \
                    (self.routes[request.URI][1] == "None" or
                     (request.COOKIES and self.routes[request.URI][1] in
request.COOKIES.keys())):
                response = self.routes[request.URI][0](request)

            elif not os.path.isfile(request.URI[1:].replace(b"%20", b" ")):
                with open(self.four_o_four, 'rb') as file:
                    response = httpro.http_message.HttpMsg(error_code=404,
                                                           headers={"content_type":
consts.MIME_TYPES[".html"]},
                                                           body=file.read())
            # If uri does not have a special path #
            else:
                # extract requested file type from URL (html, jpg etc)
                file_type = os.path.splitext(request.URI)[1]

                # generate proper HTTP header
                file_data: bytes
                with open(request.URI[1:].replace(b"%20", b" "), 'rb') as f:
```

```
                    file_data = f.read()
                response = httpro.http_message.HttpMsg(headers={"content_type":
consts.MIME_TYPES[file_type.decode()]},
                                                        body=file_data)

            consts.HTTP_LOGGER.info(f"Request to {request.URI} got response of
{response.error_code}")
            client_socket.send(response.build_message_bytes())

    # ---------- Public functions ---------- #
    def set_four_o_four(self, route: str) -> None:
        """
        Sets the route to the 404 page.
        :param route: The route to the 404 page html.
        :return: None
        """
        if isinstance(route, str):
            self.four_o_four = route
        else:
            raise TypeError("Function must get a string that holds the route to the
404 html file")

    def close_app(self) -> None:
        """
        Shuts down the server and closes the server
        :return: None
        """
        self.__closed = True
        consts.HTTP_LOGGER.debug("Closing app.")

    def run(self, port=consts.PORT, host=consts.IP) -> None:
        """
        Starts the http server.
        :return: None
        """
        global readable_socks_list, writeable_socks_list, exception_socks_list

        # Setting up the socket #
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.settimeout(.5)

        sock.bind((host, port))
        sock.listen()

        socket_list = [sock]

        try:
            while not self.__closed:
                readable_socks_list, writeable_socks_list, exception_socks_list =
select.select(socket_list,
```

```
socket_list,

socket_list,

.1)
                for notified_socket in readable_socks_list:
                    if notified_socket == sock:  # checking for new connection #
                        consts.HTTP_LOGGER.debug("Getting new connection")
                        client_socket, client_addr = sock.accept()
                        self.logger.info(consts.NEW_CLIENT.format(client_addr[0],
client_addr[1]))
                        client_socket.settimeout(.5)
                        socket_list.append(client_socket)  # Adding the socket to the
connected clients #
                    else:
                        try:
                            consts.HTTP_LOGGER.debug("Starting receive")
                            message: httpro.http_parser.HttpParser =
self.__receive_message(notified_socket)
                            consts.HTTP_LOGGER.info(b"Got Request: " + message.URI if
message.URI else "None")
                            self.__handle_client(message, notified_socket)

                        except Exception as e:
                            consts.HTTP_LOGGER.exception(e)

                        finally:
                            notified_socket.close()
                            socket_list.remove(notified_socket)
        finally:
            consts.HTTP_LOGGER.debug("Closing main socket.")
            sock.close()
            consts.HTTP_LOGGER.debug("Main socket closed.")
```

server\httpro\constants.py

```
"""
    AUTHOR: Ophir Nevo Michrowski
    DATE: 15/03/24
    DESCRIPTION: This file contains all the needed constants for the HTTP packet
"""
import logging

# HTTP necessities #
HTTP_VERSION = b"HTTP/1.1"
HEADER_SEPERATOR = b"\r\n"
BODY_SEPERATOR = HEADER_SEPERATOR * 2

# Other #
RECV_TIMEOUT = 3

# Logging #
LOG_LEVEL = self.logger.debug
LOG_DIR = r"Logs"
LOG_FORMAT = "%(asctime)s | %(levelname)s | %(message)s"
LOG_FILE_NAME = LOG_DIR + r"\http_logs.log"
HTTP_LOGGER_NAME = "http_log"
HTTP_LOGGER = logging.getLogger(HTTP_LOGGER_NAME)

# Headers #
LOCATION_HEADER = b"Location: %s"
CONTENT_TYPE_HEADER = b"Content-Type: %s"
CONTENT_LEN_HEADER = b"Content-Length: %d"
HOST_HEADER = b"Host: %s"

# Dictionaries #
REQUEST_TYPES = {
    "get": b"GET",
    "post": b"POST",
    "put": b"PUT",
    "delete": b"DELETE"
}

ERROR_CODES = {
    200: b"OK",
    500: b"INTERNAL SERVER ERROR",
    302: b"MOVED TEMPORARILY",
    403: b"FORBIDDEN",
    404: b"NOT FOUND",
    400: b"BAD REQUEST",
    301: b"MOVED PERMANENTLY",
    401: b"UNAUTHORIZED",
    405: b"METHOD NOT ALLOWED",
    413: b"PAYLOAD TOO LARGE",
    502: b"BAD GATEWAY",
    503: b"SERVICE UNAVAILABLE"
```

```
}

MIME_TYPES = {
    ".html": "text/html;charset=utf-8",
    ".jpg": "image/jpeg",
    ".css": "text/css",
    ".js": "text/javascript; charset=UTF-8",
    ".txt": "text/plain",
    ".ico": "image/x-icon",
    ".gif": "image/jpeg",
    ".png": "image/png",
    ".ttf": "font/ttf",
    "sse": "text/event-stream"
}

# Socket Related #
RECV_LENGTH = 1024
PORT = 80
IP = "0.0.0.0"

# Logging level - info #
NO_BODY = "Message has no body."
NEW_CLIENT = "{}:{} connected."
TESTS_RUN = "Auto tests were run."
FOUR_O_FOUR = "httpro/html_defaults/404.html"
```

server\httpro\functions.py

```
"""
    AUTHOR: Ophir Nevo Michrowski
    DATE: 16/03/24
    DESCRIPTION: Useful functions on dictionaries
"""


def dict_to_bytes(dictionary: dict) -> dict[bytes]:
    """
    Takes a dictionary and casts all the elements within to bytes.
    EXAMPLE: {"name": "John", "age": 45} -> {b"name": b"John", b"age": b"45"}
    :param dictionary: the dict to convert
    :return dict[bytes]: the converted dictionary .
    """
    if dictionary:
        return {(key.encode('utf-8') if isinstance(key, str) else
str(key).encode('utf-8')):
                    (value.encode('utf-8') if isinstance(value, str) else
str(value).encode('utf-8'))
                for key, value in dictionary.items()}
    else:
        return dict()
```

server\httpro\http_message.py

```
"""
    AUTHOR: Ophir Nevo Michrowski
    DATE: 15/03/24
    DESCRIPTION: Class that allows to create easy to use http_ophir messages
including responses and requests
"""
# Imports #
import httpro.constants as consts
import httpro.functions


class HttpMsg:
    """Create easy to use http messages including responses and requests"""

    def __init__(self, error_code: int = 200, body: bytes = b"", **headers) -> None:
        """
        The constructor of the HttpMsg class.
        :param error_code: The error code of the message.
        :param body: The body of the message.
        :param headers: The headers of the message - host=127.0.0.1 -> { b"host":
b"127.0.0.1"}
        :return: None
        """
        if headers is None:
            headers = dict()
        self.error_code = self.__error_code_finder(error_code)
        self.body = body
        self.headers = httpro.functions.dict_to_bytes(headers)
        if body != b"":
            self.headers[b"content_length"] = str(len(body)).encode()

    @staticmethod
    def __error_code_finder(error_code: int) -> bytes:
        """
        Returns the error code and message using the error code.
        :param error_code: The error code to be found
        :return bytes: The error code
        """
        return_value = b"-1"
        if error_code in consts.ERROR_CODES.keys():
            try:
                return_value = str(error_code).encode() + b" " +
consts.ERROR_CODES[error_code]
            except KeyError:
                consts.HTTP_LOGGER.exception(f"{error_code} is not valid")
                raise KeyError(f"{error_code} is not valid")
            except Exception:
                consts.HTTP_LOGGER.exception(f"error while finding error code")

        return return_value
```

```
    def prettify(self) -> None:
        """
        Prints the message in a readable detailed format
        :return: None
        """
        print(f"## HTTP VERSION ##\n{consts.HTTP_VERSION.decode('utf-8')}\n## ERROR
CODE ##\n" +
              f"{self.error_code.decode('utf-8')}\n## HEADERS
##\n{self.__build_headers_bytes().decode('utf-8')}\n" +
              f"## BODY ##\n{self.body.decode('utf-8')}")

    def __build_headers_bytes(self) -> bytes:
        """
        Formats the headers into bytes
        :return bytes: Byte string with the headers formatted
        """
        headers = b""
        if self.headers.items():
            for key, value in self.headers.items():
                headers += key.title().replace(b'_', b'-') + b": " + value +
consts.HEADER_SEPERATOR
        else:
            consts.HTTP_LOGGER.debug("no headers found")

        return headers

    def build_message_bytes(self) -> bytes:
        """
        Builds the http_ophir message
        :return bytes: The http_ophir message
        """
        headers = self.__build_headers_bytes()
        return consts.HTTP_VERSION + b" " + self.error_code + consts.HEADER_SEPERATOR
+ headers + \
            consts.HEADER_SEPERATOR + self.body

    def __str__(self) -> str:
        """
        Str dunder function for the HttpMsg class
        :return str: The http_ophir message in full
        """
        return self.build_message_bytes().decode('utf-8')


def auto_test_http_message() -> None:
    """
    Auto test for the HttpMsg class
    :return: None
    """
    http_request_example = b"HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\nContent-
Length: 5\r\n\r\nhello"
```

```
    http_message_test = HttpMsg(body=b"hello",
content_type=consts.MIME_TYPES[".txt"])

    assert http_message_test.build_message_bytes() == http_request_example
    assert http_message_test.headers == {b'content_type': b'text/plain',
b'content_length': b'5'}
    assert str(http_message_test) == http_request_example.decode()
    assert http_message_test.body == b"hello"
    assert http_message_test.error_code == b"200 OK"
```

server\httpro\http_parser.py

```
"""
    AUTHOR: Ophir Nevo Michrowski
    DATE: 15/03/24
    DESCRIPTION: Takes a received http_ophir request and formats it into a structure
that is easy to use in code
"""
# Imports #
import httpro.constants as consts
import httpro


# HttpParser #
class HttpParser:
    """Class to make http requests usable in code more easily"""

    def __init__(self, http_request: bytes) -> None:
        """
        The constructor of the HttpParser class.
        :param http_request: the request for the class to parse.
        :return: None
        """
        # If the request is invalid #
        is_request_valid = httpro.is_valid_request(http_request)
        if not is_request_valid["valid"] or not http_request:
            self.HTTP_REQUEST = is_request_valid["reason"]

            self.METHOD = None
            self.URI = None
            self.HTTP_VERSION = None
            self.QUERY_PARAMS = None
            self.HEADERS = None
            self.BODY = None
            self.COOKIES = None

        # If the request is valid #
        else:
            self.HTTP_REQUEST = http_request

            self.HEADERS = self.__header_parser()

            self.COOKIES = dict() if b"Cookie" in self.HEADERS.keys() else None
            if b"Cookie" in self.HEADERS.keys():
                if len(self.HTTP_REQUEST.split(b";")) > 1:
                    tmp = [x.split(b"=") for x in self.HEADERS[b"Cookie"].split(b";
")]
                    self.COOKIES = dict(tmp)
                else:
                    key, value = self.HTTP_REQUEST.split(b"=")
                    self.COOKIES[key] = value
```

```
            self.BODY = self.__body_parser()
            self.METHOD, self.URI, self.HTTP_VERSION, self.QUERY_PARAMS =
self.__section_one_parser()

        # Whether the request is valid #
        self.IS_VALID: bool = is_request_valid["valid"]

    def __header_parser(self) -> dict[bytes]:
        """
        Extracts the headers from the request
        :return dict[bytes]: {header: value}
        """
        try:
            return dict(x.split(b": ") for x in
self.HTTP_REQUEST.split(consts.HEADER_SEPERATOR)[1:-2])
        except IndexError:
            consts.HTTP_LOGGER.debug("Request has no headers.")

    def __body_parser(self) -> bytes:
        """
        Extracts the body from the requests
        :return bytes: the body of the request
        """
        return self.HTTP_REQUEST.split(consts.BODY_SEPERATOR)[1]

    def __section_one_parser(self) -> tuple[bytes, bytes, bytes, dict[bytes] or
None]:
        """
        Splits the first section of the http_ophir request (before the headers)
        :return tuple: (method, uri, http_version, query_params)
        """
        request = self.HTTP_REQUEST.split(b' ')

        def __query_params_parser() -> dict[bytes] or None:
            """
            Extract the parameters from the http_ophir request
            :return: dictionary of all the query parameters
            """
            try:
                return dict(
                    x.split(b" ")[0].split(b"=", 1)[0:2] for x in
request[1].split(b"?", 1)[1].split(b"&", 1))

            # If there are no params #
            except IndexError:
                consts.HTTP_LOGGER.error("no query parameters")
                return None
            except Exception as e:
                consts.HTTP_LOGGER.exception(e)

        return request[0], request[1], request[2], __query_params_parser()
```

```
    def __str__(self) -> str:
        """
        Str dunder function for the HttpMsg class
        :return str: The http_ophir message in full
        """
        return self.HTTP_REQUEST.decode('utf-8')


def auto_test_http_parser() -> None:
    """
    Automatic tests for the HttpParser class.
    :return: None
    """
    # Valid HTTP request with query parameters
    valid_request = b"GET /index.html?param1=value1&param2=value2 HTTP/1.1\r\nHost:
example.com\r\n\r\n"
    parser_valid = HttpParser(valid_request)
    assert parser_valid.IS_VALID
    assert parser_valid.HTTP_REQUEST == valid_request
    assert parser_valid.QUERY_PARAMS == {b"param1": b"value1", b"param2": b"value2"}
    assert parser_valid.HEADERS == {b"Host": b"example.com"}
    assert parser_valid.BODY == b""

    # Invalid HTTP request
    parser_invalid = HttpParser(b"GET /index.html HTTP/1.1\r\nHost:
example.com\r\nInvalid-Header\r\n\r\n")
    assert not parser_invalid.IS_VALID
    assert not parser_invalid.QUERY_PARAMS
    assert not parser_invalid.HEADERS
    assert not parser_invalid.BODY
```

server\server.py

```
"""
    main file of the transmitting client
"""
import threading

from transmitting_client_handler.tc_handle import start_tc_handle


def main() -> None:
    """
    Starts streaming
    :return: None
    """
    tc_thread = threading.Thread(target=start_tc_handle)

    tc_thread.start()

    tc_thread.join()


if __name__ == '__main__':
    main()
```

server\transcriptor\transcript.py

```
import json
import vosk
from src.transmitting_client.audio_capture import AudioCapture
from src.server.utils.logger import Logger

class Transcriptor:
    """
    Handles live speech recognition and captioning using AudioCapture.
    """

    def __init__(self):
        """
        Initializes the speech-to-text engine using Vosk and sets up audio capture.
        """
        self.model = vosk.Model(r"C:\Users\ophir\Downloads\vosk-model-small-en-us-
0.15")  # Make sure you have a Vosk model downloaded
        self.recognizer = vosk.KaldiRecognizer(self.model, 16000)  # Match sample
rate
        self.logger = Logger("Transcriptor").logger

        # Initialize direct audio capture
        self.audio_capture = AudioCapture(self.logger)

    def transcribe_audio(self) -> None:
        """
        Continuously captures and transcribes live audio.
        :return: None
        """
        self.logger.debug("Transcribing live audio... Speak now!")

        while True:
            try:
                audio_data = self.audio_capture.get_audio_chunk()
                if not audio_data:
                    continue

                # Process audio with Vosk
                if self.recognizer.AcceptWaveform(audio_data):
                    result = json.loads(self.recognizer.Result())
                    self.logger.debug(f"Caption: {result['text']}")  # Display live
transcription

            except KeyboardInterrupt:
                self.logger.debug("Stopping transcription...")
                break

        self.audio_capture.release()```

## `server\transmitting_client_handler\audio_receive_handler.py`
```

```python
"""
    Audio Handling
"""
# Imports #
import numpy as np
import pyaudio
import time

from src.server.transmitting_client_handler.udp_handler_generic import
UDPServerHandler
from src.server.utils.consts.tc_consts import Ports, AudioConsts


class AudioReceiveHandler:
    """
    Handles real-time UDP audio reception with delta time synchronization.
    """

    @staticmethod
    def recv_audio() -> None:
        """
        Receives RTP audio packets and plays audio in real-time using delta time for
better timing.
        :return: None
        """
        udp_handler = UDPServerHandler(Ports.AUDIO_PORT)  # Use correct port

        audio = pyaudio.PyAudio()
        stream = audio.open(format=AudioConsts.FORMAT,
                            channels=AudioConsts.CHANNELS,
                            rate=AudioConsts.SAMPLE_RATE,
                            output=True,
                            frames_per_buffer=AudioConsts.CHUNK_SIZE)

        print("Receiving Audio... Press Ctrl+C to stop.")

        prev_time = time.time()  # Track last packet timestamp
        expected_interval = 0.02  # 20ms expected interval for real-time audio

        while True:
            try:
                audio_data = udp_handler.receive_rtp_message()
                if not audio_data:
                    continue

                current_time = time.time()
                delta_time = current_time - prev_time  # Time difference since last
packet

                # Adjust playback timing dynamically
                if delta_time < expected_interval:
```

```
                    time.sleep(expected_interval - delta_time)  # Compensate for
timing drift

                # Convert received data into NumPy array
                audio_array = np.frombuffer(audio_data, dtype=np.int16)

                # Play received audio
                stream.write(audio_array.tobytes())

                prev_time = current_time  # Update last received packet time

            except KeyboardInterrupt:
                break

        # Cleanup
        stream.stop_stream()
        stream.close()
        audio.terminate()
        print("Audio reception stopped.")


if __name__ == '__main__':
    AudioReceiveHandler.recv_audio()
```

server\transmitting_client_handler\rtp_parser.py

```
"""
    This file contains the RTPPacketDecoder class, which handles incoming RTP packets
and parses them into a usable object.
"""
# Imports #
import struct
from src.server.utils.logger import Logger


class RTPPacketDecoder:
    """
    Decodes RTP packets into header fields and payload.
    """

    def __init__(self, packet: bytes) -> None:
        """
        Initializes the RTPPacketDecoder instance with the given RTP packet.

        :param packet: (bytes) The raw RTP packet.
        """
        self.packet = packet
        self.version = None
        self.padding = None
        self.extension = None
        self.csrc_count = None
        self.marker = None
        self.payload_type = None
        self.sequence_number = None
        self.timestamp = None
        self.ssrc = None
        self.csrcs = []
        self.extension_data = None
        self.payload = None

        self.logger = Logger("rtp-logger").logger

        self.decode_packet()

    def decode_packet(self) -> None:
        """
        Decodes the RTP packet, extracting its header and payload.
        :return: None
        """
        try:
            if len(self.packet) < 12:
                raise ValueError("RTP packet too short to contain a valid header.")

            # Parse the fixed header (12 bytes)
            header = struct.unpack('!BBHII', self.packet[:12])
            self.version = (header[0] >> 6) & 0b11
```

```
            self.padding = (header[0] >> 5) & 0b1
            self.extension = (header[0] >> 4) & 0b1
            self.csrc_count = header[0] & 0b1111
            self.marker = (header[1] >> 7) & 0b1
            self.payload_type = header[1] & 0b01111111
            self.sequence_number = header[2]
            self.timestamp = header[3]
            self.ssrc = header[4]

            # Calculate header size and extract CSRCs if present
            header_size = 12
            if self.csrc_count > 0:
                csrc_end = header_size + 4 * self.csrc_count
                if len(self.packet) < csrc_end:
                    raise ValueError("RTP packet too short to contain all CSRCs.")
                self.csrcs = struct.unpack(f'!{self.csrc_count}I',
self.packet[header_size:csrc_end])
                header_size = csrc_end

            # Handle extension header if present
            if self.extension:
                if len(self.packet) < header_size + 4:
                    raise ValueError("RTP packet too short to contain the extension
header.")
                ext_header = struct.unpack('!HH', self.packet[header_size:header_size
+ 4])
                ext_length = ext_header[1] * 4  # Length in 32-bit words
                ext_end = header_size + 8 + ext_length
                if len(self.packet) < ext_end:
                    raise ValueError("RTP packet too short to contain extension
data.")
                self.extension_data = self.packet[header_size + 8:ext_end]
                header_size = ext_end

            # Extract payload
            self.payload = self.packet[header_size:]

            self.logger.info("RTP packet successfully decoded.")

        except struct.error as e:
            self.logger.exception("Failed to unpack RTP packet: %s", e)
            raise ValueError("Invalid RTP packet structure.") from e
        except Exception as e:
            self.logger.exception("Error decoding RTP packet: %s", e)
            raise

    def get_header_info(self) -> dict:
        """
        Returns the RTP header fields as a dictionary.

        :return: dict: Header fields with their values.
        """
```

```
        return {
            "version": self.version,
            "padding": self.padding,
            "extension": self.extension,
            "csrc_count": self.csrc_count,
            "marker": self.marker,
            "payload_type": self.payload_type,
            "sequence_number": self.sequence_number,
            "timestamp": self.timestamp,
            "ssrc": self.ssrc,
            "csrcs": self.csrcs,
            "extension_data": self.extension_data,
        }

    def get_payload(self) -> bytes:
        """
        Returns the payload from the RTP packet.

        :return: bytes: The raw payload data.
        """
        return self.payload
```

server\transmitting_client_handler\tc_handle.py

```
"""
    main file of the transmitting client
"""
import threading

from audio_receive_handler import AudioReceiveHandler
from video_receive_handler import VideoReceiveHandler


def start_tc_handle() -> None:
    """
    Starts streaming
    :return: None
    """
    # Create processes
    video_thread = threading.Thread(target=VideoReceiveHandler.recv_video())
    audio_thread = threading.Thread(target=AudioReceiveHandler.recv_audio())

    # Start processes
    video_thread.start()
    audio_thread.start()

    # Keep the main program running
    video_thread.join()
    audio_thread.join()
```

server\transmitting_client_handler\udp_handler.py

```
"""
    Will hold the class in charge of communications
"""
# Imports #
import logging
import zlib
import socket

import struct
import cv2
import numpy as np

from src.server.transmitting_client_handler.rtp_parser import RTPPacketDecoder
from src.server.utils.consts.logging_consts import *
from src.server.utils.consts.tc_consts import CommunicationConsts
from src.server.utils.logger import Logger


class UDPServerHandler:
    """
    This class represents the video receiving server using UDP.
    It initializes a UDP socket to listen for incoming packets and
    uses RTPacketDecoder to decode the received RTP packets.
    """

    def __init__(self):
        """
        Initializes the Server instance.

        :return: None
        """
        self.bind_address = CommunicationConsts.HOST
        self.bind_port = CommunicationConsts.PORT
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind((self.bind_address, self.bind_port))

        self._uncompleted_frame_packets = dict()

        self.sock.settimeout(CommunicationConsts.FRAGMENT_RECEIVE_TIMEOUT)

        self.logger = Logger("UDP-logger").logger

        self.logger.info(SuccessMessages.SERVER_LISTENING.format(self.bind_address,
self.bind_port))

    def _receive_packet(self, buffer: int) -> bytes | None:
        """
        Will receive a packet using a UDP connection
        :param buffer: (int) the buffer to receive
        :return bytes | None: The received packet if one was received
```

```
        """
        packet = b''
        try:
            packet, _ = self.sock.recvfrom(buffer)
        except socket.timeout:
            self.logger.debug("Got a None packet")
            return None
        except socket.error as e:
            self.logger.exception("NETWORK ERROR: ", e)

        return packet

    def assemble_frame(self, seq_start: int, seq_end: int) -> bytes | None:
        """
        Assembles / Discards frames
        :param seq_start: (int) fragment sequence start
        :param seq_end: (int) fragment sequence end
        :return bytes | None: bytes if a frame was assembled, None if frame was
dropped
        """
        frame = b''
        for seq in range(seq_start, seq_end + 1):
            packet = self._uncompleted_frame_packets.get(seq)
            if not packet:
                self._uncompleted_frame_packets = dict()
                return None
            assert isinstance(packet, RTPPacketDecoder)

            frame += packet.payload
        self._uncompleted_frame_packets = dict()

        return zlib.decompress(frame)

    def receive_video(self):
        """
        Receives RTP packets from the client, decodes them, and displays the video
frames.
        """
        fps = 0
        import time
        start = time.time()
        while True:
            try:
                packet = self._receive_packet(CommunicationConsts.BUFFER_SIZE)
                if not packet:
                    self.logger.debug("Got a None Packet")
                    continue
                else:
                    decoded_packet = RTPPacketDecoder(packet)
                    self.logger.debug("Got a good packet")

                if len(self._uncompleted_frame_packets) > 0 and \
```

```
next(iter(self._uncompleted_frame_packets.values())).timestamp <
decoded_packet.timestamp:
                    self._uncompleted_frame_packets = dict()

                self._uncompleted_frame_packets[decoded_packet.sequence_number] =
decoded_packet

                if decoded_packet.marker !=
CommunicationConsts.EXPECT_ANOTHER_FRAGMENT:
                    seq_start = decoded_packet.sequence_number - \
                                struct.unpack('!I', decoded_packet.extension_data)[0]
+ 1

                    frame_data = self.assemble_frame(seq_start,
decoded_packet.sequence_number)
                    if frame_data is None:
                        continue

                    array = np.frombuffer(frame_data, dtype=np.uint8)
                    # frame = array.reshape((480, 640, 3))
                    frame = np.resize(array, (240, 320, 3))

                    frame = cv2.resize(frame, (640, 480))

                    # Display the frame using OpenCV
                    cv2.imshow("Received Video", frame)

                    # Count fps #
                    fps += 1
                    if time.time() - start >= 1:
                        self.logger.info(f"FPS: {fps}")
                        fps = 0
                        start = time.time()

                    # Break the loop if 'q' is pressed
                    if cv2.waitKey(1) & 0xFF == ord('q'):
                        break

            except socket.error as e:
                self.logger.exception(ErrorMessages.VIDEO_RECEIVING_ERROR, e)
                break

        # Cleanup OpenCV windows after exiting the loop
        cv2.destroyAllWindows()


if __name__ == '__main__':
    UDPServerHandler().receive_video()
```

server\transmitting_client_handler\udp_handler_generic.py

```
"""
    Will hold the class in charge of communications
"""

# Imports #
import zlib
import socket
import struct

from src.server.transmitting_client_handler.rtp_parser import RTPPacketDecoder
from src.server.utils.consts.logging_consts import *
from src.server.utils.consts.tc_consts import CommunicationConsts, Ports
from src.server.utils.logger import Logger


class UDPServerHandler:
    """
    This class represents the video receiving server using UDP.
    It initializes a UDP socket to listen for incoming packets and
    uses RTPPacketDecoder to decode the received RTP packets.
    """

    def __init__(self, port: Ports):
        """
        Initializes the Server instance.

        :return: None
        """
        self.bind_address = CommunicationConsts.HOST
        self.bind_port = port.value
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind((self.bind_address, self.bind_port))

        self._uncompleted_frame_packets = dict()

        self.sock.settimeout(CommunicationConsts.FRAGMENT_RECEIVE_TIMEOUT)

        self.logger = Logger("UDP-logger").logger

        self.logger.info(SuccessMessages.SERVER_LISTENING.format(self.bind_address,
self.bind_port))

    def _receive_packet(self, buffer: int) -> bytes | None:
        """
        Will receive a packet using a UDP connection
        :param buffer: (int) the buffer to receive
        :return bytes | None: The received packet if one was received
        """
        packet = b''
        try:
```

```
            packet, _ = self.sock.recvfrom(buffer)
        except socket.timeout:
            self.logger.debug("Got a None packet")
            return None
        except socket.error as e:
            self.logger.exception("NETWORK ERROR: ", e)

        return packet

    def assemble_rtp_message(self, seq_start: int, seq_end: int) -> bytes | None:
        """
        Assembles / Discards frames
        :param seq_start: (int) fragment sequence start
        :param seq_end: (int) fragment sequence end
        :return bytes | None: bytes if a frame was assembled, None if frame was
dropped
        """
        frame = b''
        for seq in range(seq_start, seq_end + 1):
            packet = self._uncompleted_frame_packets.get(seq)
            if not packet:
                self._uncompleted_frame_packets = dict()
                return None
            assert isinstance(packet, RTPPacketDecoder)

            frame += packet.payload
        self._uncompleted_frame_packets = dict()

        return zlib.decompress(frame)

    def receive_rtp_message(self) -> bytes or None:
        """
        Receives RTP packets from the client & decodes them.
        :return bytes or None: If a message/packet is received then the payload will
be returned.
        """
        message_data: None or bytes = None
        while True:
            try:
                packet = self._receive_packet(CommunicationConsts.BUFFER_SIZE)
                if not packet:
                    self.logger.debug("Got a None Packet")
                    continue
                else:
                    decoded_packet = RTPPacketDecoder(packet)
                    self.logger.debug("Got a good packet")

                if len(self._uncompleted_frame_packets) > 0 and \

next(iter(self._uncompleted_frame_packets.values())).timestamp <
decoded_packet.timestamp:
                    self._uncompleted_frame_packets = dict()
```

```
                self._uncompleted_frame_packets[decoded_packet.sequence_number] =
decoded_packet

                if decoded_packet.marker !=
CommunicationConsts.EXPECT_ANOTHER_FRAGMENT:
                    seq_start = decoded_packet.sequence_number - \
                                struct.unpack('!I', decoded_packet.extension_data)[0]
+ 1

                    message_data = self.assemble_rtp_message(seq_start,
decoded_packet.sequence_number)
                    if message_data:
                        break

            except socket.error as e:
                self.logger.exception(ErrorMessages.VIDEO_RECEIVING_ERROR, e)
                break

        return message_data
```

server\transmitting_client_handler\video_receive_handler.py

```
"""
    This file holds the VideoReceiveHandler class
"""
# Imports #
import cv2
import numpy as np

from src.server.transmitting_client_handler.udp_handler_generic import
UDPServerHandler
from src.server.utils.consts.tc_consts import Ports


class VideoReceiveHandler:
    """
    Handles real-time UDP video reception
    """

    @staticmethod
    def recv_video() -> None:
        """
        Receives video and presents it
        :return: None
        """
        udp_handler = UDPServerHandler(Ports.VIDEO_PORT)

        while True:
            frame = udp_handler.receive_rtp_message()
            array = np.frombuffer(frame, dtype=np.uint8)
            # frame = array.reshape((480, 640, 3))
            frame = np.resize(array, (240, 320, 3))

            frame = cv2.resize(frame, (640, 480))

            # Display the frame using OpenCV
            cv2.imshow("Received Video", frame)

            # Break the loop if 'q' is pressed
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        # Cleanup OpenCV windows after exiting the loop
        cv2.destroyAllWindows()


if __name__ == '__main__':
    VideoReceiveHandler.recv_video()
```

server\utils\consts\logging_consts.py

```
"""
    Global Logging Messages for the server
"""
# Imports #
import logging


class LoggerConsts:
    FORMAT = "%(asctime)s - %(levelname)s - %(message)s"
    LOG_LEVEL = logging.DEBUG
    LOG_DIR = r"../logs/"
    LOG_FILE_EXTENSION = ".log"


class SuccessMessages:
    """
    Enum to define success log messages.
    """
    SERVER_LISTENING = "Server is now listening on {}:{}"
    PACKET_SENT = "RTP packet successfully sent to the server."
    PACKET_RECEIVED = "RTP packet successfully received by the server."
    VIDEO_STREAM_STARTED = "Video streaming has started successfully."
    CONNECTION_ESTABLISHED = "Connection successfully established with {}:{}"


class ErrorMessages:
    """
    Enum to define error log messages.
    """
    VIDEO_TRANSMISSION_ERROR = "An error occurred while transmitting video: {}"
    VIDEO_RECEIVING_ERROR = "An error occurred while receiving video: {}"
    DECODE_PACKET_ERROR = "Failed to decode the RTP packet: {}"
    PAYLOAD_EXTRACTION_ERROR = "Error extracting payload from the RTP packet: {}"
    SOCKET_CREATION_ERROR = "Error creating the UDP socket: {}"
    CONNECTION_ERROR = "Error while establishing connection to the server: {}"
    VIDEO_STREAM_ERROR = "Error occurred during video streaming: {}"
```

server\utils\consts\tc_consts.py

```
"""
    Holds the consts for the udp server
"""
from enum import Enum

from pyaudio import paInt16


# Communication Consts #
class CommunicationConsts:
    PORT = 5004
    HOST = '127.0.0.1'
    PAYLOAD_TYPE = 32  # uncompressed video streams
    BUFFER_SIZE = 65535  # Max UDP packet
    EXPECT_ANOTHER_FRAGMENT = 0
    FRAGMENT_RECEIVE_TIMEOUT = .1  # In Seconds


class Ports(Enum):
    """
    All available Ports
    """
    VIDEO_PORT = 5004
    AUDIO_PORT = 5006


# Audio Consts #
class AudioConsts:
    CHANNELS = 1
    SAMPLE_RATE = 44100
    CHUNK_SIZE = 512
    FORMAT = paInt16
```

server\utils\logger.py

```
"""
    Holds a generic logger class to handle logs.
"""
# Imports #
import logging
import os

from src.server.utils.consts.logging_consts import LoggerConsts


class Logger:
    """
    Handles all logs
    """

    def __init__(self, logger_name: str) -> None:
        """
        Initialize logger
        :param logger_name: (str) the name of the logger
        :return: None
        """
        self.format = LoggerConsts.FORMAT
        self.level = LoggerConsts.LOG_LEVEL
        self.file_name = logger_name + LoggerConsts.LOG_FILE_EXTENSION
        self.file_path = LoggerConsts.LOG_DIR + self.file_name

        if not os.path.isdir(LoggerConsts.LOG_DIR):
            os.makedirs(LoggerConsts.LOG_DIR)
            print("created dir")

        self.logger = logging.getLogger(logger_name)

        self.logger.setLevel(self.level)
        file_handler = logging.FileHandler(self.file_path)
        file_handler.setFormatter(logging.Formatter(self.format))
        self.logger.addHandler(file_handler)
```

transmitting_client\audio_capture.py

```
"""
    This file contains the AudioCapture class
"""
# Imports #
from utils.consts import AudioCaptureConsts
import pyaudio
import logging


class AudioCapture:
    """ Handles real-time audio capture and streaming """

    def __init__(self, logger: logging.Logger) -> None:
        """
        Initializes the audio capture instance.
        :return: None
        """
        self.rate = AudioCaptureConsts.SAMPLE_RATE
        self.channels = AudioCaptureConsts.CHANNELS
        self.chunk_size = AudioCaptureConsts.CHUNK_SIZE

        self.logger = logger

        try:
            self.audio = pyaudio.PyAudio()
            self.stream = self.audio.open(format=pyaudio.paInt16,
                                          channels=self.channels,
                                          rate=self.rate,
                                          input=True,
                                          frames_per_buffer=self.chunk_size)
            self.logger.info("Audio capture initialized successfully.")
        except Exception as e:
            self.logger.exception(f"Error initializing audio capture: {e}")
            raise

    def get_audio_chunk(self) -> bytes:
        """
        Captures and returns a single chunk of audio data.

        :return bytes: Raw audio data.
        """
        try:
            data = self.stream.read(self.chunk_size, exception_on_overflow=False)
            return data
        except Exception as e:
            self.logger.exception(f"Error capturing audio chunk: {e}")
            return b""

    def release(self) -> None:
        """
```

```
        Cleans up resources.
        :return: None
        """
        self.stream.stop_stream()
        self.stream.close()
        self.audio.terminate()
        self.logger.info("Audio capture released.")
```

transmitting_client\audio_transmission_handler.py

```
import time
from audio_capture import AudioCapture
from rtp_handler import RTPHandler
from udp_handler import UDPClientHandler
from utils.consts import Ports
from utils.logger import Logger
from utils.payload_types import PayloadTypes


class AudioTransmissionHandler:
    """
    Handles real-time audio transmission via RTP over UDP with delta time
synchronization.
    """

    def __init__(self):
        """
        Initializes the full audio streaming pipeline.
        """
        self.logger = Logger("audio-logger").logger
        self.audio_capture = AudioCapture(self.logger)
        self.rtp_handler = RTPHandler(PayloadTypes.AUDIO)
        self.udp_handler = UDPClientHandler(Ports.AUDIO_PORT.value)

        self.logger.info("AudioTransmissionHandler initialized.")

    def start_streaming(self) -> None:
        """
        Begins live audio transmission with delta time control for better
synchronization.
        :return: None
        """
        self.logger.info("Starting live audio stream...")

        prev_send_time = time.time()  # Track last packet send time
        send_interval = 0.02  # 20ms expected interval for real-time audio

        while True:
            try:
                # Capture live audio chunk
                audio_chunk = self.audio_capture.get_audio_chunk()

                # Wrap chunk in RTP packets
                rtp_packets = self.rtp_handler.create_packets(audio_chunk)

                # Compute delta time
                current_time = time.time()
                delta_time = current_time - prev_send_time

                # Adjust sending rate dynamically
```

```
                if delta_time < send_interval:
                    time.sleep(send_interval - delta_time)  # Smooth out transmission
timing

                # Send packets via UDP
                self.udp_handler.send_packets(rtp_packets)

                prev_send_time = current_time  # Update last sent timestamp

            except Exception as e:
                self.logger.exception(f"Error during audio transmission: {e}")
                break

        self.stop_streaming()

    def stop_streaming(self) -> None:
        """
        Cleans up resources after transmission stops.
        :return: None
        """
        self.logger.info("Stopping audio transmission...")
        self.audio_capture.release()


if __name__ == '__main__':
    AudioTransmissionHandler().start_streaming()
```

transmitting_client\rtp_handler.py

```
"""
    This file holds the RTP handler class -
"""
# Imports #
import math
import random
import struct
import time
import zlib

from utils import consts
from utils.payload_types import PayloadTypes
from utils.logger import Logger


class RTPHandler:
    """
    This class handles the RTP protocol, including creating and parsing RTP packets.
    """

    def __init__(self, payload_type: PayloadTypes, start_timestamp=(int(time.time() *
1000) % (2 ** 32))) -> None:
        """
        Initializes the RTPHandler instance.

        :param start_timestamp: (int) The start_timestamp for the object
        :param payload_type: (PayloadTypes) The payload type for the RTP stream.
        :return: None
        """

        self.payload_type = payload_type.value
        self.ssrc = random.randint(0, 2 ** 32 - 1)  # Generate a random 32-bit SSRC
        self.sequence_number = 0
        self.timestamp = start_timestamp
        self._update_timestamp()

        self.logger = Logger("rtp-logger").logger

    def build_header(self, marker: int = 0, csrcs: list[int] = None, extension_data:
bytes = None) -> bytes:
        """
        Constructs the RTP header.

        :param extension_data:
        :param marker: (int) The marker bit.
        :param csrcs: (list[int]) List of contributing source identifiers.
        :return: bytes: The RTP header as a byte string.
        """
        if csrcs is None:
            csrcs = []
```

```
        cc = len(csrcs)
        version = 2
        padding = 0
        extension = extension_data is not None
        self.sequence_number += 1

        header = (
                (version << 30) |
                (padding << 29) |
                (extension << 28) |
                (cc << 24) |
                (marker << 23) |
                (self.payload_type << 16) |
                (self.sequence_number & 0xFFFF)
        )
        header_bytes = struct.pack('!II', header, self.timestamp)
        ssrc_bytes = struct.pack('!I', self.ssrc)
        csrc_bytes = b''.join(struct.pack('!I', csrc) for csrc in csrcs)
        extension_bytes = b''
        if extension_data:
            extension_profile_id =
consts.CommunicationConsts.RTP_EXTENSION_PROFILE_ID
            extension_length = struct.pack('!I', math.ceil(len(extension_data) /
4))[2:]
            extension_header = consts.CommunicationConsts.RTP_EXTENSION_HEADER
            extension_bytes = extension_profile_id + extension_length +
extension_header + extension_data

        return header_bytes + ssrc_bytes + csrc_bytes + extension_bytes

    def create_packets(self, payload: bytes ,csrcs: list[int] = None) -> list[bytes]:
        """
        Creates an RTP packet by combining the header and payload.

        :param payload: (bytes) payload to put in the rtp packet
        :param csrcs: (list[int]) List of contributing source identifiers.
        :return: bytes: The complete RTP packet.
        """
        try:
            payload = zlib.compress(payload)

            self._update_timestamp()

            payloads = []
            payload_pointer = 0
            header_len = len(self.build_header(0, csrcs,
extension_data=struct.pack('!I', len(payloads))))
            while payload_pointer < len(payload):
                payload_read_end_index = payload_pointer +
consts.CommunicationConsts.MAX_UDP_PAYLOAD_SIZE - header_len
                # payload_read_end_index = payload_pointer + 1500 - header_len
```

```
                payloads.append(payload[payload_pointer:payload_read_end_index])
                payload_pointer = payload_read_end_index

            ext_data = struct.pack('!I', len(payloads))

            packets = []
            for i in range(len(payloads)):
                is_last_frag = i == len(payloads) - 1
                header = self.build_header(int(is_last_frag), csrcs,
extension_data=ext_data)
                packets.append(header + payloads[i])

            self.logger.info("RTP packets created successfully.")

        except Exception as e:
            self.logger.exception("Error while creating RTP packet: %s", e)
            raise

        return packets

    def get_ssrc(self) -> int:
        """
        Returns the SSRC for the RTP stream.

        :return: int: The SSRC value.
        """
        return self.ssrc

    def _update_timestamp(self) -> None:
        """
        updates timestamp
        :return: None
        """
        self.timestamp = int(time.time() * 1000) % (2 ** 32)  # Initialize with
current time in milliseconds & 32 bit
```

transmitting_client\tc_main.py

```
"""
    main file of the transmitting client
"""
import multiprocessing

from audio_transmission_handler import AudioTransmissionHandler
from video_transmission_handler import VideoTransmission


def main() -> None:
    """
    Starts streaming
    :return: None
    """
    # Create processes
    video_process =
multiprocessing.Process(target=VideoTransmission().transmit_video)
    audio_process =
multiprocessing.Process(target=AudioTransmissionHandler().start_streaming)

    # Start processes
    video_process.start()
    audio_process.start()

    # Keep the main program running
    video_process.join()
    audio_process.join()


if __name__ == '__main__':
    main()
```

transmitting_client\udp_handler.py

```
"""
    Will hold the class incharge of communications
"""
# Imports #
import logging
import socket
import time

from utils.consts import CommunicationConsts
from utils.logging_messages import *
from utils.logger import Logger


class UDPClientHandler:
    """
        This class represents the video transmitting client using UDP.
        It initializes an RTPHandler instance to handle RTP packet creation
        and uses a UDP socket for sending packets.
    """

    def __init__(self, port: int):
        """
        Initializes the Client instance.

        :param port: (int) the port to send the data through
        :return: None
        """
        self.server_address = CommunicationConsts.HOST
        self.server_port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

        self.logger = Logger("UDP-logger").logger

    def send_packets(self, packets: list[bytes]) -> bool:
        """
        Captures video frames, creates RTP packets, and transmits them to the server
via UDP.
        :param packets: (list[bytes]) packets to send
        :return bool: whether operation was successful
        """
        success = True
        try:
            # Send the packet to the server
            self.logger.debug("Sending frame, fragmented into {}
packets".format(len(packets)))
            for packet in packets:
                self.sock.sendto(packet, (self.server_address, self.server_port))
            self.logger.info(SuccessMessages.PACKET_SENT)

        except Exception as e:
```

```
            self.logger.exception(ErrorMessages.VIDEO_TRANSMISSION_ERROR, e)
            success = False

        return success
```

transmitting_client\utils\consts.py

```
"""
    This file holds all the constant variables of the transmitting client
"""
# Imports #
import logging

import pyaudio
from enum import Enum


class Ports(Enum):
    """
    All available Ports
    """
    VIDEO_PORT = 5004
    AUDIO_PORT = 5006

class CommunicationConsts:
    HOST = '127.0.0.1'

    MAX_UDP_PAYLOAD_SIZE = 65507
    RTP_HEADER_SIZE = 12
    MAX_RTP_PAYLOAD_SIZE = MAX_UDP_PAYLOAD_SIZE - RTP_HEADER_SIZE
    RTP_EXTENSION_PROFILE_ID = b'00'
    RTP_EXTENSION_HEADER = b'0000'


class AudioCaptureConsts:
    CHANNELS = 1
    SAMPLE_RATE = 44100
    CHUNK_SIZE = 512
    FORMAT = pyaudio.paInt16


class LoggerConsts:
    FORMAT = "%(asctime)s - %(levelname)s - %(message)s"
    LOG_LEVEL = logging.DEBUG
    LOG_DIR = r"logs/"
    LOG_FILE_EXTENSION = ".log"
```

transmitting_client\utils\logger.py

```
"""
    Holds a generic logger class to handle logs.
"""
# Imports #
import logging
import os

from src.transmitting_client.utils.consts import LoggerConsts


class Logger:
    """
    Handles all logs
    """

    def __init__(self, logger_name: str) -> None:
        """
        Initialize logger
        :param logger_name: (str) the name of the logger
        :return: None
        """
        self.format = LoggerConsts.FORMAT
        self.level = LoggerConsts.LOG_LEVEL
        self.file_name = logger_name + LoggerConsts.LOG_FILE_EXTENSION
        self.file_path = LoggerConsts.LOG_DIR + self.file_name

        if not os.path.isdir(LoggerConsts.LOG_DIR):
            os.makedirs(LoggerConsts.LOG_DIR)
            print("created dir")

        self.logger = logging.getLogger(logger_name)

        self.logger.setLevel(self.level)
        file_handler = logging.FileHandler(self.file_path)
        file_handler.setFormatter(logging.Formatter(self.format))
        self.logger.addHandler(file_handler)
```

transmitting_client\utils\logging_messages.py

```
"""
    Global Logging Messages for the transmitting client
"""
class ErrorMessages:
    """
    Constants to define error log messages.
    """
    OPEN_VIDEO_SOURCE = "Unable to open video source: %s"
```

```
    VIDEO_CAPTURE_INIT = "Error initializing video capture for source {source}:
{error}"

    RETRIEVE_FRAME = "Failed to retrieve frame."
    GET_FRAME_FROM_SOURCE = "Error retrieving frame from video source {source}:
{error}"

    RELEASE_SOURCE = "Error releasing video capture for source {source}: {error}"

    VIDEO_TRANSMISSION_ERROR = "Error while transmitting video: %s"


class SuccessMessages:
    """
    Constants to define success log messages.
    """
    RETRIEVE_FRAME = "Frame retrieved successfully."
    RELEASE_VIDEO_SOURCE = "Video source %s released successfully."
    VIDEO_CAPTURE_INIT = "Video Capture initiated successfully"

    PACKET_SENT = "Packet sent to server"
```

transmitting_client\utils\payload_types.py

```
"""
    RTP payload types
"""
# Imports #
from enum import Enum

class PayloadTypes(Enum):
    VIDEO = 0
    AUDIO = 0```

## `transmitting_client\video_capture.py`

```python
"""
    This file holds the VideoCapture class
"""

import logging

# Imports #
import cv2
import numpy as np

from src.transmitting_client.utils.logging_messages import ErrorMessages,
SuccessMessages


class VideoCapture:
    """
    This class handles the video capture and handling.
    """

    def __init__(self, logger: logging.Logger, source=0) -> None:
        """
        Initializes the video capture instance.

        :param logger: (Logger) logger for this class
        :param source: (int or str): The video source, can be a camera index or a
video file path.
        :return: None
        """
        self.logger = logger
        self.source = source

        try:
            self.capture = cv2.VideoCapture(self.source)
            if not self.capture.isOpened():
                self.logger.error(ErrorMessages.OPEN_VIDEO_SOURCE % self.source)
                raise ValueError(ErrorMessages.OPEN_VIDEO_SOURCE % self.source)
```
```

```
            self.logger.info(SuccessMessages.VIDEO_CAPTURE_INIT)

        except Exception as e:

self.logger.exception(ErrorMessages.VIDEO_CAPTURE_INIT.format(source=self.source,
error=e))
            raise

    def get_frame(self):
        """
        Retrieves a single frame from the video source.

        :return tuple: (success, frame), where `success` is a boolean indicating if
the frame was read successfully, and `frame` is the captured frame.
        """
        res_tuple: tuple[bool, np.ndarray]
        try:
            success, frame = self.capture.read()
            if success:
                self.logger.debug(SuccessMessages.RETRIEVE_FRAME)

                new_size = (frame.shape[1] // 2, frame.shape[0] // 2)
                img_resized = cv2.resize(frame, new_size,
interpolation=cv2.INTER_AREA)
                frame = np.array(img_resized)

            else:
                self.logger.error(ErrorMessages.RETRIEVE_FRAME)
            return success, frame

        except Exception as e:

self.logger.exception(ErrorMessages.GET_FRAME_FROM_SOURCE.format(source=self.source,
error=e))
            res_tuple = (False, np.ndarray(shape=(0, 0)))

        return res_tuple

    def release(self) -> None:
        """
        Releases the video capture resource.

        :return: None
        """
        try:
            if self.capture:
                self.capture.release()
                self.logger.info(SuccessMessages.RELEASE_VIDEO_SOURCE % self.source)

        except Exception as e:
```

```
self.logger.exception(ErrorMessages.RELEASE_SOURCE.format(source=self.source,
error=e))

    def __del__(self) -> None:
        """
        Ensures resources are released when the instance is deleted.

        :return: None
        """
        self.release()
        self.logger.info(SuccessMessages.RELEASE_VIDEO_SOURCE % self.source)
```

transmitting_client\video_transmission_handler.py

```
"""
    This file holds the VideoTransmission class.
"""
# Imports #
import logging
import time

import numpy as np

from rtp_handler import RTPHandler
from udp_handler import UDPClientHandler
from utils import consts
from utils.payload_types import PayloadTypes
from video_capture import VideoCapture
from utils.logger import Logger


class VideoTransmission:
    """
    Bundles video handlers and handles transmission
    """

    def __init__(self, video_capture_source: int = 0):
        """
        Initiating class members

        :param video_capture_source: (int) camera port
        """
        self.port = consts.Ports.VIDEO_PORT.value
        self.video_capture_source = video_capture_source
        self.payload_type = PayloadTypes.VIDEO

        self.rtp_handle = RTPHandler(self.payload_type)
        self.udp_handle = UDPClientHandler(self.port)

        self.logger = Logger("video-logger").logger

    def transmit_video(self) -> None:
        """
        transmit video using rtp
        :return: None
        """
        video_capture = VideoCapture(self.logger, self.video_capture_source)

        while True:
            # Capture start time
            start_time = time.time()

            success, frame = video_capture.get_frame()
            if success and isinstance(frame, np.ndarray):
```

```
            payload = frame.tobytes()
            packets: list[bytes] = self.rtp_handle.create_packets(payload)

            _ = self.udp_handle.send_packets(packets)

        # Handling delta time
        delta_time = time.time() - start_time
        time.sleep(max(0.033 - delta_time, 0))  # ~30 frames per second
```